REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	~~06/14/~~97	Final 01Jan94 - 31Dec96

4. TITLE AND SUBTITLE
Development ~~of~~ Applications of New Algorithms for the Simulation of Viscous Compressible Flows with Moving Bodies

5. FUNDING NUMBERS
F49620-94-1-0119

6. AUTHOR(S)
Ronald Lohner, Chi Yang, Jaime Cebral

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
George Mason U

8. PERFORMING ORGANIZATION
AFOSR-TR-97
0160

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office Of Scientific Research
Aerospace & Materials Sciences Directorate
110 Duncan Avenue, Suite B-115
Bolling AFB DC 20332-0001
NA

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
94-1-0119

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE
DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The overall objective of the research carried out over the last two years was the development of new algorithms for the efficient simulation of viscous compressible flows with moving bodies in three dimensions using unstructured grids. The development was based on current 3-D Euler/Navier-Stokes capabilities, and encompassed flow solvers, grid generation. and the efficient use of emerging supercomputer hardware. The research carried out over the last three years significantly advanced the state of the art in this area of CFD. The particular topics are treated below in detail.

14. SUBJECT TERMS
CFD, Viscous flow, Unstructured

15. NUMBER OF PAGES
210

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

FINAL REPORT: DECEMBER 1996

DEVELOPMENT AND APPLICATION OF NEW ALGORITHMS FOR THE SIMULATION OF VISCOUS COMPRESSIBLE FLOWS WITH MOVING BODIES IN THREE DIMENSIONS

Rainald Löhner, Chi Yang and Juan R. Cebral
GMU/CSI, The George Mason University
Fairfax, VA 22030-4444

SUMMARY

The overall objective of the research carried out over the last two years was the development of new algorithms for the efficient simulation of viscous compressible flows with moving bodies in three dimensions using unstructured grids. The development was based on current 3-D Euler/Navier-Stokes capabilities, and encompassed flow solvers, grid generation, and the efficient use of emerging supercomputer hardware. The research carried out over the last three years significantly advanced the state of the art in this area of CFD. The particular topics are treated below in detail.

1. FLOW SOLVERS

For the flow solvers, seven major developments took place over the course of this research effort:

a) Implicit flow solvers;
b) Improved spatial discretization and boundary conditions;
c) Better mesh moving strategies;
d) Optimal, vectorized interpolation schemes;
e) Parallel h-refinement;
f) Link to CSD Codes strategy;
g) Validation studies; and
h) Store ejection from a hypersonic plane.

1.1 Implicit flow solvers

Implicit flow solvers are considered essential for the efficient simulation of viscous, compressible, time-dependent flows. We developed a linearized implicit scheme that uses a Generalized Minimal RESiduals algorithm in conjunction with incomplete lower-upper (ILU) preconditioning for the solution of the Euler and Navier-Stokes equations. The results were encouraging, showing that for Euler problems steady state results could be achieved in less than 40 steps [1,2]. On the other hand, the storage costs and the cost of getting close to the solution at the start of the iteration were considered suboptimal. For technical details the reader is referred to [1], which is reproduced in Appendix 1.

1.2 Improved spatial discretization and boundary conditions

Upon comparison of several high order schemes [3], improvements were made to a node-centered upwind finite volume scheme for the solution of the Euler and Navier-Stokes equations on unstructured meshes. The improvements included a more accurate boundary integration procedure, which was consistent with the finite element approximation, and a new reconstruction scheme based on the consistent mass matrix iteration. The numerical results indicated that the present scheme significantly improved the quality of numerical solutions with very little additional computational cost. For technical details the reader is referred to [4], which is reproduced in Appendix 2.

1.3 Better mesh moving strategies

A Laplacian smoothing of the mesh velocities with variable diffusivity based on the distance from moving bodies was developed [6-8]. This variable diffusivity enforces a more uniform mesh velocity in the region close to the moving bodies. Given that in most applications these are regions where small elements are located, the new procedure decreases considerably element distortion, reducing the need for local or global remeshing, and in some cases avoiding it alltogether. A hypersonic store release was used to test the new algorithm. Numerical results obtained show that the new mesh

velocity smoothing leads to a much less deformed grid close to the moving missile. For this case, the number of local remeshings required dropped by a factor of 1:4, leading to considerable CPU savings in multiprocessor environment. Since then, this algorithm has been used extensively for many applications ([8,11,12]). For technical details the reader is referred to [7], which is reproduced in Appendix 3.

1.4 Optimal, vectorized interpolation schemes

When performing a local or global remeshing, the variables need to be interpolated from the old grid to the new one. This is typically done using a scalar fast neighbour search. We monitored this process on the CRAY-C90 and found, to our surprise, that it took a considerable CPU time. Therefore, we vectorized the interpolation procedure. The speed-ups obtained ranged from 1:4.5-1:5.0 on a 1-processor system as compared to the best, optimized scalar code. This led to a considerable reduction of CPU times. For technical details the reader is referred to [9], which is reproduced in Appendix 4.

1.5 Parallel H-Refinement

The classic h-refinement was extended to MIMD parallel machines. The main innovation consisted of new data structures to handle the compatibility of refinement and de-refinement cases allowed [10]. For this first demonstration, several important algorithmic aspects, such as the subsequent parallel load balancing, were left out. The basic idea, though, proved its worth. We feel that more work is required to complete this effort.

1.6 Link to CSD Codes

In order to solve, in a cost-effective manner, fluid-structure interaction problems, a loosely coupled algorithm to combine computational fluid dynamics (CFD) and computational structural dynamics (CSD) was devised. In this algorithm, the structure is used as the 'master-surface' to define the extent of the fluid region, and the fluid is used as the 'master-surface' to define the loads. The transfer of loads, displacements, and velocities is carried out via fast interpolation and projection algorithms. This fluid-structure algorithm can be interpreted as an iterative solution to the fully coupled, large matrix problem that results from the discretization of the complete problem. The advantage of this new algorithm is that it allows a cost effective re-use of existing software, with minimum amount of alterations required to account for the interaction of the different media.

Several example runs using FEFLO96 as the CFD code, and DYNA3D as the CSD code, demonstrate the effectiveness of the proposed methodology. For more details, see the AIAA invited paper, Ref. [11], which is reproduced in Appendix 5, as well as [12],[13]. The load transfer was made conservative, a characteristic that sets it apart from all transfer algorithms used to date [13].

1.7 Validation studies

The efficiency and fidelity of the new Arbitrary Lagrangian-Eulerian (ALE) methodology on unstructured grids was validated by two simulations. This validation effort was part of an ongoing research effort to develop a cost-efficient and accurate numerical methodology capable of simulating the motion of complex-geometry, three-dimensional bodies embedded in external, temporally and spatially evolving flow-fields.

The first computation modeled the release of a finned store from a generic wing/pylon configuration. The numerical Eulerian predictions were compared to the available experimental data for both the wing and the separating store placed at three drop distances. Very good agreement was obtained between the predicted and measured Cp axial variation on the wing and along four angular cuts on the captive store.

The second simulation modeled multiple-store ejection from an F-117 fighter. The efficiency of this simulation was aided tremendously by the improved local remeshing methodology. Although the stores traveled a very long distance, only ten global remeshings were required, compared to approximately sixty to eighty that would have been required with the old methodology. The replacement of the expensive global remeshing, is the key to the affordability of such a large-scale moving body simulations. For more details, see Ref. [6,8].

1.8 Store ejection from a hypersonic plane

In order to investigate the interactions of missiles and planes during the store ejection at hypersonic speeds, a numerical simulation model was built. This model separates the whole ejection process into three stages. The first one is the steady motion part, where the plane flies at the Mach number of 8 and there is no relative motions among missles and the plane. The second one is the prescribed motion part, where the plane continues flying at the same Mach number, and each missile moves in terms of its prescribed relative motion. The third one is the free motion part, where the plane continues flying at the same Mach number, and the motion of the missiles are solely controled by aerodynamic forces. Three missiles with two in the front and one in the back are used in the simulation. The missiles starts turning after they are released.

This simulation is very CPU intensive, and requires a lot of local and global remeshings. This simulation was made possible by the new ALE mesh velocity for moving body was developed to reduce the development of the deformed grids close to the moving missiles. This technique is described above in the section 1.3. Numerical results obtained show that the number of local remeshings required dropped by a factor of 1:4, leading to considerable CPU savings in a multiprocessor environment. The average element number was about 750,000 during the run. A global remeshing was required about every 2,000 timesteps. A film of this simulation was made on a SGI workstation and delivered to the contract monitor.

2. GRID GENERATION

In the area of grid generation, there were five major developments that took place during the course of this research effort:

a) Improvements in robustness and speed;
b) Surface meshing from discrete data;
c) Element size attached to CAD-data;
d) Adaptive background grids; and
e) Navier-Stokes gridding.

2.1 Improvements in robustness and speed

At the beginning of this effort, the advancing front technique was still prone to occasional failures, and relatively slow. With the advent of more powerful supercomputers, the problem complexity increased, leading to larger grids. These large grids accentuated the remaining problems in mesh generation. The robustness of the advancing front technique was enhanced by only allowing the creation of well-formed elements, additional neighbour checks, transformation to unit frame, checks for the usage of close points and faces, and the introduction of additional distance criteria for front-crossing tests. Speed was enhanced by storing (and interpolating) background grid and source data not at the faces (as previously done), but at points. For large number of sources, this can lead to speed-up factors of 1:5.

2.2 Surface meshing from discrete data

An advancing front surface gridding technique that operates on discretely defined faces was developed [14,15]. This technique is based on three steps: surface feature recovery, actual gridding, and surface recovery. The following aspects have to be considered carefully in order to make the precedure reliable for complex geometries:

a) Recovery of surface features and discrete surface patches from the discrete data,
b) Filtering based on point and side normals to remove undesirable data close to cusps and corners,
c) Proper choice of host faces for ridges, and
d) Fast interpolation procedures suitable for complex geometry.

Several examples ranging from academic to industrial demonstrated the utility of the developed procedure for ab initio surface meshing from discrete data, such as encountered when the surface description is already given as discrete, the improvement of existing surface triangulations, as well as remeshing applications during runs exhibiting significant change of domain. For technical details the reader is referred to [15], which is reproduced in Appendix 6.

2.3 Element size attached to CAD-data

For problems that require gridding complex geometries, the specification of proper element sizes can become a tedious process. Conventional background grids would involve many tetrahedra, whose generation is a labor-intensive, tedious task. Point, line, or surface-sources are not always appropriate either. A better way to address these problems was devised by attaching element size directly to CAD-data. For many problems, the smallest elements are required close to the boundary. Therefore, the next element size may be obtained by multiplying it with a user-specified increase factor if the element size for the points of the current front is stored. See Refs. [16-18] for more details.

2.4 Adaptive background grids

In order to reduce the amount of user intervention to a minimum, we developed adaptive background grid refinement. We defined where to refine and how to refine. The refinement was made in two passes: Pass 1: background grid adjustment, and Pass 2: selection of elements to be refined. The new designed background grid adaption may be used to automatically generate grids that represent the surface within a required or prescribed accuracy. See Refs. [16-18] for more details.

2.5 Navier-Stokes gridding

Creating highly stretched grids of acceptable quality for complex configurations has been an outstanding goal for over two decades. We developed a technique that separates the zones to be meshed into mainly isotropic (Euler region) and mainly anisotropic (RANS region, close to walls/shear layers). The RANS regions are meshed first, by growing prismatic elements from the boundary. These prismatic elements are subdivided into tetrahedra. The resulting mesh is analyzed for element shape and size. Bad/large/distorted elements are removed accordingly. This results in a first front of faces. The remainder of the domain is gridded using the traditional advancing front technique. This procedure works well [16-19], but is not completely general. We have, as of late, tried to mesh some very complex geometries with it, and have encountered difficulties. Some possible solutions habe been proposed but not yet implemented. For technical details the reader is referred to [18], which is reproduced in Appendix 7.

3. EFFICIENT USE OF SUPERCOMPUTING HARDWARE

In order to keep up with the rapid advance of supercomputing hardware, all algorithms used were examined with respect to their suitability for distributed memory parallel machines. There were two major developments that took place during the course of this research effort in this area:

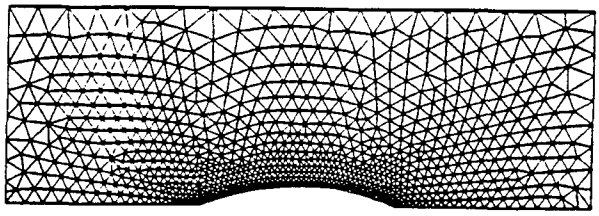
a) Load balancing;
b) Timing and benchmarking.

3.1 Load balancing

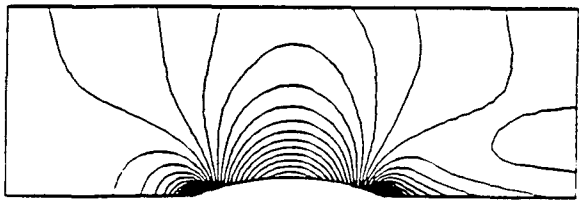
A new load balancing scheme, based on a greedy algorithm with diffusion-based improvement was developed and tested. The algorithm allows for almost perfect (<1%) load balance for arbitrary number of processors, load per element and mesh topology in less than 20 passes over the mesh [20]. This is in contrast to popular recursive subdivision techniques, where the number of processors must be a power of 2, and local element imbalances are more difficult to account for.

3.2 Timings and benchmarking

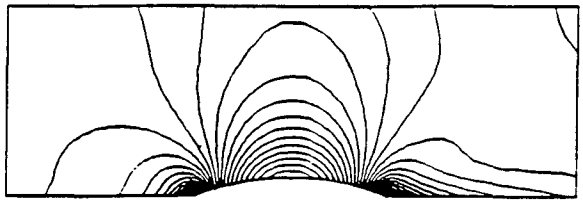
The new load balancing scheme was used to split meshes and time them on several parallel platforms, including the Intel Touchstone and Paragon, Thinking Machines CM5, and IBM SP2. The timings were restricted to steady-state problems, i.e. no moving/deforming meshes were present, and no h-refinement was used. For this class of applications, the timings showed almost perfect speed-up, even for very large number of processors (>380 for the Intel Touchstone). These results are very encouraging and bode well for the future of the schemes used so far. Work is continuing in order to port all aspects of the methodology to a distributed memory parallel hardware environment.



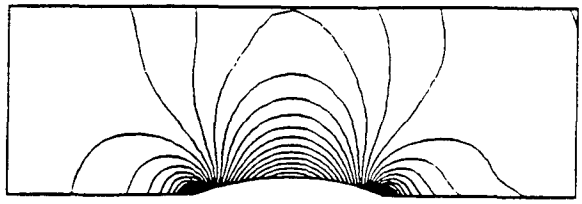
Computational mesh



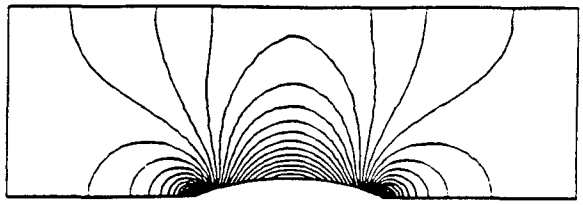
Velocity contours by Green-Gauss



Velocity contours by least-squares



Velocity contours by consistent mass



Velocity contours by potential

Fig2a. mesh and results for channel problem

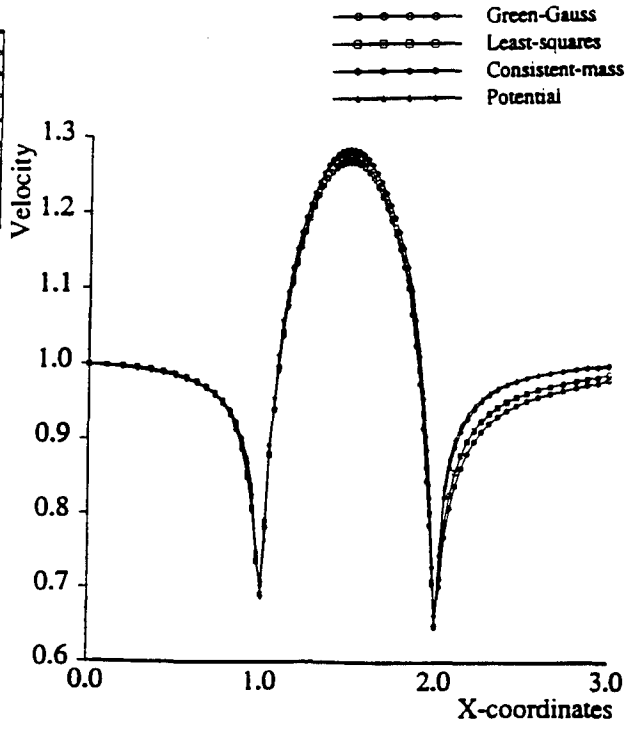


Fig.2b Comparison of velocity distribution

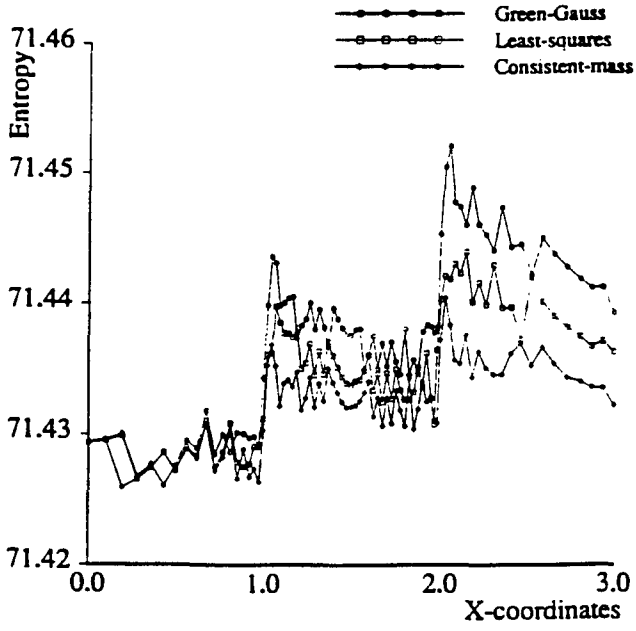


Fig.2c Comparison of entropy distribution

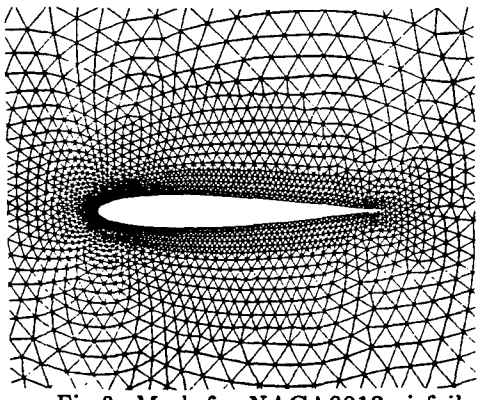


Fig.3a Mesh for NACA0012 airfoil

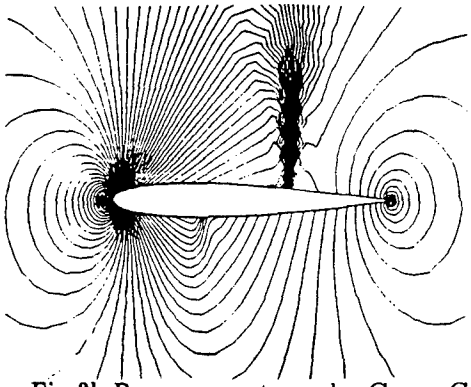


Fig.3b Pressure contours by Green-Gauss

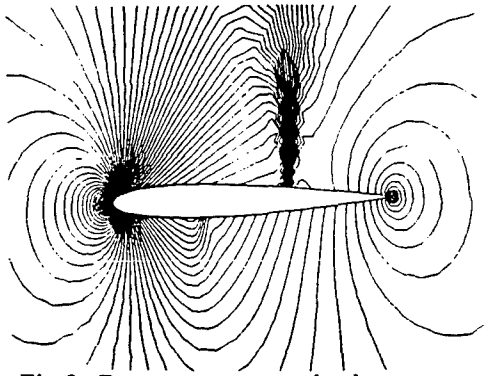


Fig.3c Pressure contours by least-squares

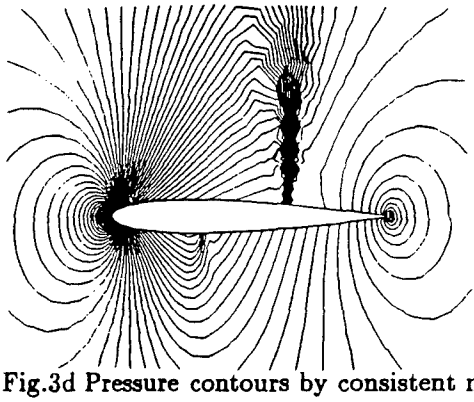


Fig.3d Pressure contours by consistent mass

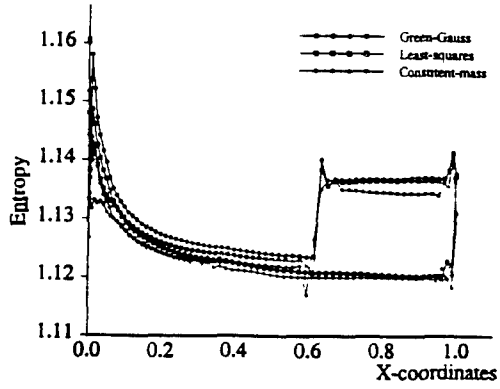


Fig.3e Comparison of entropy distribution

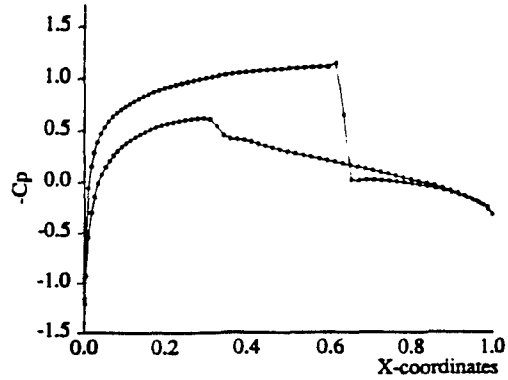


Fig.3f C_p distribution by Green-Gauss

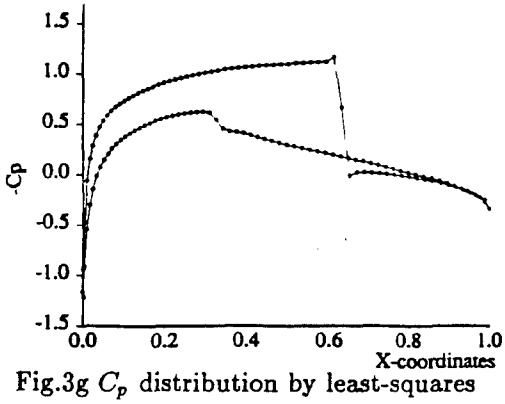


Fig.3g C_p distribution by least-squares

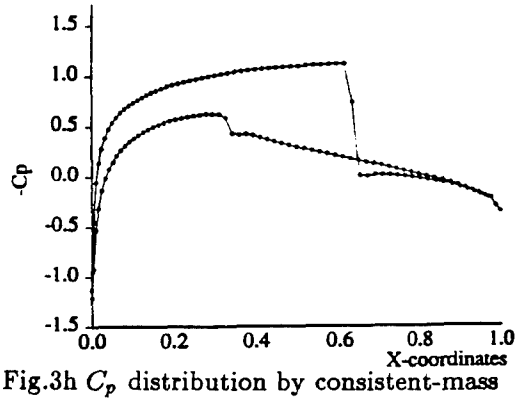


Fig.3h C_p distribution by consistent-mass

4. SUMMARY

The present research effort significantly advanced the state of the art in the simulation of compressible viscous flows with moving bodies. A number of breakthroughs and 'firsts' were achieved, of which the following are considered the most important:

- The first simulation using an implicit scheme with unstructured grids with more than a million elements [1];
- An optimal mesh velocity technique based on a nonlinear Laplacian that minimizes remeshing requirements by an order of magnitude [7];
- The first vectorized, optimal-speed interpolation techniques for unstructured grids [9];
- The first conservative load transfer algorithm for fluid-structure interaction simulations [13];
- The first simulation of compressible flows with more than a hundred independently moving bodies [8];
- The most complex fluid-structure interaction simulation to date [12]; and
- The first realistic CFD simulation with close to 400 processors [21].

More work is still required to transform these algorithms into daily production tools that can be used effectively in the design and engineering process.

5. REFERENCES

We published extensively in the literature. Some of these papers are reproduced in the sequel.

5.1 Flow Solvers, Implicit:

[1] H. Luo, J.D. Baum, R. Löhner and J. Cabello - Implicit Finite Element Schemes and Boundary Conditions for Compressible Flows on Unstructured Grids; *AIAA*-94-0816 (1994).

[2] J. Cabello, K. Morgan, R. Löhner, H. Luo and J.D. Baum - An Implicit Solver for Laminar Compressible Flows on Unstructured Grids; *AIAA*-95-0344 (1995).

5.2 Flow Solvers, Discretization Schemes:

[3] J. Cabello, K. Morgan and R. Löhner - A Comparison of Higher Order Schemes Used in a Finite Volume Solver for Unstructured Grids; *AIAA*-94-2293 (1994).

[4] H. Luo, J.D. Baum and R. Löhner - An Improved Finite Volume Scheme for Compressible Flows on Unstructured Grids; *AIAA*-95-0348 (1995).

5.3 Flow Solvers, ALE Methodology:

[5] J.D. Baum, H. Luo and R. Löhner - A New ALE Adaptive Unstructured Methodology for the Simulation of Moving Bodies; *AIAA*-94-0414 (1994).

[6] J.D. Baum, H. Luo and R. Löhner - Validation of a New ALE, Adaptive Unstructured Moving Body Methodology for Multi-Store Ejection Simulations; *AIAA*-95-1792 (1995).

[7] R. Löhner and Chi Yang - Improved ALE Mesh Velocities for Moving Bodies; *Comm. Num. Meth. Eng.* 12, 599-608 (1996).

[8] R. Löhner, Chi Yang and J.D. Baum - Rigid and Flexible Store Separation Simulations Using Adaptive Unstructured Grid Technologies; pp.1-29 in *Proc. 1st AFOSR Conf. on Dynamic Motion CFD* (L. Sakell and D.D. Knight eds.), Rutgers University, New Brunswick, New Jersey, June (1996).

5.4 Flow Solvers, Reinterpolation:

[9] R. Löhner - Robust, Vectorized Search Algorithms for Interpolation on Unstructured Grids; *J. Comp. Phys.* 118, 380-387 (1995).

5.5 Flow Solvers, H-Refinement:

[10] A. Shostko and R. Löhner - Parallel 3-D H-Refinement; *AIAA*-95-1662-CP (1995).

5.6 Link to CSD:

[11] R. Löhner, C. Yang, J. Cebral, J.D. Baum, H. Luo, D. Pelessone and C. Charman - Fluid-Structure Interaction Using a Loose Coupling Algorithm and Adaptive Unstructured Grids; *AIAA*-95-2259 [Invited] (1995).

[12] J.D. Baum, H. Luo, R. Löhner, C. Yang, D. Pelessone and C. Charman - A Coupled Fluid/Structure Modeling of Shock Interaction with a Truck; *AIAA*-96-0795 (1996).

[13] J.R. Cebral and R. Löhner - Conservative Load Projection and Tracking for Fluid-Structure Problems; *AIAA*-96-0797 (1996).

5.7 Grid Generation:

[14] R. Löhner - Surface Meshing from Discrete Data; Paper presented at the 4th International Meshing Roundtable, Albuquerque, NM, October (1995).

[15] R. Löhner - Regridding Surface Triangulations; *J. Comp. Phys.* 126, 1-10 (1996).

[16] R. Löhner - Extending the Range of Applicability and Automation of the Advancing Front Grid Generation Technique; *AIAA*-96-0033 (1996).

[17] R. Löhner - Extensions and Improvements of the Advancing Front Grid Generation Technique; *Comm. Num. Meth. Eng.* 12, 683-702 (1996).

[18] R. Löhner - Progress in Grid Generation via the Advancing Front Technique; *Engineering with Computers* 12, 186-210 (1996).

[19] R. Löhner - Matching Semi-Structured and Unstructured Grids for Navier-Stokes Calculations; *AIAA*-93-3348-CP (1993).

5.8 Supercomputing

[20] R. Löhner and R. Ramamurti - A Load Balancing Algorithm for Unstructured Grids; *Comp. Fluid Dyn.* 5, 39-58 (1995).

[21] R. Ramamurti and R. Löhner - A Parallel Implicit Incompressible Flow Solver Using Unstructured Meshes; *Computers and Fluids* 5, 119-132 (1996).

APPENDIX 1: IMPLICIT FLOW SOLVERS

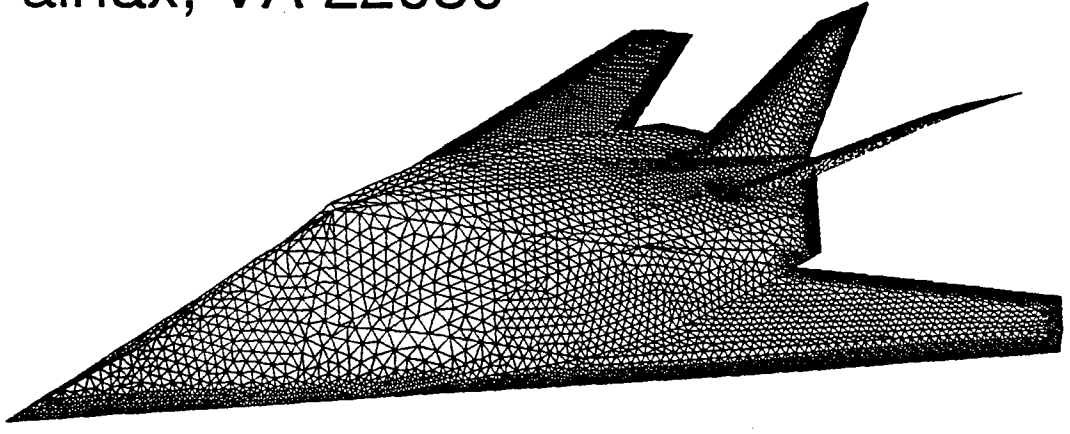
AIAA 94-0816

Implicit Schemes and Boundary Conditions for Compressible Flows on Unstructured Meshes

Hong Luo*, Joseph D. Baum*, Rainald Löhner**, and Jean Cabello**

* Science Applications International Corporation,
McLean, VA 22102

** George Mason University
Fairfax, VA 22030



32nd Aerospace Sciences Meeting & Exhibit

January 10-13, 1994 / Reno, NV

where V_i is the volume of the dual mesh cell (equivalent to the lumped mass matrix in the finite element), R_i is the right-hand-side residual and equals to zero for a steady state solution. The Euler implicit discretization and linearization of equation (4.1) in time leads to a system of linear equations

$$(\frac{V}{\Delta t} + \frac{\partial R}{\partial U})^n \Delta U^n = R^n \qquad (4.2)$$

where Δt is the time increment, and ΔU^n the difference of unknown vector between time levels n and $n+1$, i.e.,

$$\Delta U^n = U^{n+1} - U^n \ . \qquad (4.3)$$

Note that as Δt tends to infinity, the scheme reduces to standard Newton's method for solving a system of nonlinear equations. Newton's method is known to have quadratic convergence property. The term $\frac{\partial R}{\partial U}$ represents symbolically the Jacobian matrix. It involves the linearization of the numerical flux vectors. For example, considering Roe's approximation of the inviscid flux vector:

$$R_{inv}(U_i, U_j, n) = \frac{1}{2}(F(U_i, n) + F(U_j, n)) - \frac{1}{2} \mid A(\tilde{U}) \mid (U_j - U_i) \ , \qquad (4.4)$$

the Jacobian terms $\frac{\partial R_{inv}}{\partial U_i}$ and $\frac{\partial R_{inv}}{\partial U_j}$ must be derived. The exact Jacobian linearization of Roe's flux function is possible, but extremely expensive to evaluate. A good approximation, which is computationally efficient, is to neglect the terms arising from differentiation of $\mid A(\tilde{U}) \mid$ in the linearization process. This leads to the following approximate linearization

$$\frac{\partial R_{inv}}{\partial U_i} = \frac{1}{2}\frac{\partial F(U_i, n)}{\partial U_i} + \frac{1}{2} \mid A(\tilde{U}) \mid = \frac{1}{2}(A(U_i) + \mid A(\tilde{U}) \mid) \qquad (4.5)$$

$$\frac{\partial R_{inv}}{\partial U_j} = \frac{1}{2}\frac{\partial F(U_j, n)}{\partial U_j} - \frac{1}{2} \mid A(\tilde{U}) \mid = \frac{1}{2}(A(U_j) - \mid A(\tilde{U}) \mid). \qquad (4.6)$$

The justification of this approximation can be found in reference[11]. In addition, in order to reduce the number of non-zero elements in the matrix and to simplify the linearization, only a first order representation of the inviscid flux terms is linearized. This results in the graph of the sparse matrix $\frac{\partial R}{\partial U}$ being identical to the graph of the supporting unstructured mesh. The penalty in making these approximations in the linearization process is that the quadratic convergence of Newton's method can never be achieved because of the mismatch between the right and left hand side operators in equation (4.2). The viscous terms are linearized in a straightforward manner, except that the viscosity coefficient is not linearized.

Equation (4.2) represents a system of linear simultaneous algebraic equations and needs to be solved at each time step. This system of equations can be solved by direct matrix inversion; however, this requires considerable computer memory and CPU time. Iterative methods are attractive due to their computational efficiency and relatively low memory requirements. In this work, the system of linear equations is solved by the Generalized Minimal RESidual (GMRES) method of Saad and Schultz[12]. It is a generalization of conjugate gradient method for solving a linear system

$$Ax = b \qquad (4.7)$$

where the coefficient matrix is not symmetric and/or positive definite. The use of GMRES combined with different preconditioning techniques is becoming widespread in the CFD community for the solution of the Euler and Navier-Stokes equations[6,10,13]. GMRES minimizes the norm of the computed residual vector $r^n = b - Ax^n$ over the subspace spanned by a certain number of orthogonal search directions. GMRES by itself is not a very efficient scheme. It must be augmented by a preconditioner to produce acceptable efficiency. It is well known that the speed of convergence of an iterative algorithm for a linear system depends on the condition number of the matrix A. The preconditioning technique involves solving an equivalent preconditioned linear system

$$\tilde{A}\tilde{x} = \tilde{b} \qquad (4.8)$$

instead of the original system (4.7), in the hope that $\tilde{A}$ is well conditioned. Three forms of preconditioners can be defined as following

$$P^{-1}Ax = P^{-1}b \ , \qquad (4.9)$$

$$AQQ^{-1}x = b \ , \qquad (4.10)$$

and

$$P^{-1}AQQ^{-1}x = P^{-1}b \ . \qquad (4.11)$$

The systems of linear equations in equations (4.9), (4.10), and (4.11) are referred to, respectively, as left-preconditioned, right-preconditioned, and symmetric-preconditioned, and P and Q as left and right preconditioners.

The motivation for preconditioning is twofold: a) reduce the computational effort required to solve the linearized system of equations at each time-step, and b) reduce the total number of time-steps required to obtain a steady state solution. Preconditioning will be cost-effective only if the additional computational work incurred for each sub-iteration is compensated for by a reduction in the total number of iterations to convergence, so that the total cost of solving the overall non-linear system is reduced. In the present work, a preconditioner derived from the block incomplete lower-upper factorization of matrix A has been found to be an effective preconditioner and has been

used throughout. All left, right, and symmetric preconditionings have been implemented.

Grid renumbering is also used to improve the convergence of the GMRES. Ordering of nodes in the grid has been found to affect the convergence rate of iterative solvers. Following reference[7], the mesh is renumbered according to the Reverse Cuthill-McKee method.

Using an edge-based data structure, the implicit coefficient matrix is stored in upper, lower, and diagonal matrix forms. It requires a storage of $2 \times nedge \times neqns \times neqns + npoin \times neqns \times neqns$, where *npoin* is the number of grid points, *neqns* number of unknown variables, and *nedge* number of edges. The same amount of memory requirements is needed to store the preconditioning matrix. In addition, a storage corresponding to $npoin+(2\times nedge+npoin)$ is required for the two index arrays, which are necessary for the factorization of ILU. The need for additional storage associated with the GMRES algorithm is an array of size $(k+2) \times neqns \times npoin$, where k is the number of search directions. For $k = 10$, this results in about $775 * npoin$ storage locations in 3D. Compared with $95 * npoin$ storage locations needed by its explicit counterpart, the present implicit scheme requires about 8 times more memory.

5. IMPLICIT BOUNDARY CONDITIONS

The results of this investigation indicate that the treatment of boundary conditions is crucial to the success of an implicit scheme. When boundary conditions are treated explicitly, only a very limited CFL number can be used, resulting in an inefficient algorithm. In order for the implicit scheme to be stable at high CFL numbers, boundary conditions must be incorporated implicitly. In the present work, this is realized by imposing the boundary conditions during the evaluation of flux at boundary surfaces, and then by linearizing these boundary conditions and adding them to the implicit coefficient matrix. Two basic types of boundary conditions are defined: a solid wall boundary condition and inflow and outflow boundary conditions.

On the solid wall, the slip boundary conditions are assumed for inviscid flow. The tangency flow condition is implemented by imposing no flux through the wall, so that the inviscid flux normal to the boundary face is

$$\int_{\partial C_I \cap \Gamma_w} F^j n_j d\Gamma = \int_{\partial C_I \cap \Gamma_w} \begin{pmatrix} 0 \\ pn_x \\ pn_y \\ pn_z \\ 0 \end{pmatrix} d\Gamma \ . \quad (5.1)$$

For viscous flow, the no-slip boundary condition is assumed and the given wall temperature is imposed strongly.

At inflow and outflow boundaries, the flow is supposed to be advection dominated, and a precise set of compatible exterior data has to be selected, depending on the flow regime and the velocity direction. For this purpose, a plus-minus flux splitting is applied to exterior and interior values in reference[5]. More precisely, the inviscid flux at the boundaries is evaluated using Steger-Warming's flux vector splitting

$$\int_{\partial C_I \cap \Gamma_\infty} F^j n_j d\Gamma = \int_{\partial C_I \cap \Gamma_\infty} (A^+(U_i, n)U_i + A^-(U_\infty, n)U_\infty) d\Gamma \quad (5.2)$$

It is worth noting that the treatment of boundary conditions implies that for supersonic inflow, the flux imposed is computed from the fluid state at the infinity, and for supersonic outflow, the flux imposed is computed from the state variables at the cell I. Clearly, such treatment of boundary conditions is consistent with the mathematical theory of characteristics and correctly accounts for wave propagation in the far field. However, when this boundary condition was applied to the present implicit scheme, the convergence history was not satisfactory for some test cases. Hence, Steger-Warming's flux vector splitting is replaced by Roe's flux-difference splitting in this work. This leads to

$$\int_{\partial C_I \cap \Gamma_\infty} F^j n_j d\Gamma = \int_{\partial C_I \cap \Gamma_\infty} (\frac{1}{2}(F(U_i, n) + F(U_\infty, n)) - \frac{1}{2} \mid A(\tilde{U}) \mid (U_\infty - U_i)) d\Gamma \ . \quad (5.3)$$

This procedure provides a boundary point treatment that is completely compatible and consistent with the interior point differencing scheme. The numerical experience indicates that this treatment of boundary conditions gives very satisfactory convergence.

All boundary conditions are then linearized consistently, and are included in the left-hand-side matrix. For viscous flow, as Dirichlet boundary conditions are imposed on the solid wall, both left-hand-side matrix and right-hand-side vector have to be modified in order for the Dirichlet boundary conditions to be satisfied.

It is worth noting that the boundary conditions are imposed on the boundary faces, not at the boundary points. Therefore, this avoids ambiguity for a boundary point that lies on a junction between two boundary condition types.

6. NUMERICAL EXAMPLES

A variety of test cases for a wide range of flow conditions, from subsonic to supersonic, in both 2D and 3D, is selected to demonstrate the effectiveness of the present implicit scheme over its explicit counterpart. All of the computations are done using a non-restarted GMRES with a left block ILU preconditioner, unless otherwise stated. The solution tolerance for GMRES is set to 0.1 with 10 search directions. All computations were initiated with the freestream flow as the initial guess. It has been found necessary to use a small CFL number when an initial guess is taken to be the free stream condition. The start-up CFL number was around 2 and was allowed

to increase inversely proportional to the L_2 norm of the residual, up to some maximum CFL number.

A standard and widely accepted approach to judge the performance of different numerical schemes is to determine the amount of computational or CPU time required by the particular scheme to reduce the L_2 norm of the residual vector to a certain order of magnitude. However, the CPU time of an algorithm can be heavily influenced by the skills of the individual programmer. Efficient implementations often require investments in 'real' time and patience by the programmer. In addition, an implementation on one particular machine may run faster or slower on another machine, i.e., the CPU time may be machine dependent. Therefore, in this research, the numerical schemes performance was compared based on the number of time steps required to reduce the L_2 norm of the residual vector (normalized by the norm of the initial residual vector) of the problem by certain orders-of-magnitude.

Test case 1. 2D inviscid flow past NACA0012 airfoil

The problem under consideration is an inviscid transonic flow around a NACA0012 airfoil with a freestream Mach number of 0.85 and an angle of attack of 1 degree. This is a classical and significant test problem for Euler solvers. The mesh, consisting of 6,397 elements and 3,274 is shown in Fig. 1.a. Fig.1b displays the computed pressure contours in the flow field. The pressure coefficient distribution on the airfoil is shown in Fig.1c. Fig.1d displays a comparison of convergence histories among the explicit scheme, the explicit scheme with implicit residual smoothing, and the implicit scheme with ILU preconditioner, respectively. The explicit scheme results were obtained using two stage Runge-Kutta scheme and a CFL number of 0.8. The explicit scheme with implicit residual smoothing uses three stage Runge-Kutta scheme and a CFL number of 4.0. The implicit scheme results were obtained using a maximum CFL number of 10,000.

Test case 2. 2D viscous flow past a flat plate

This test case involves a laminar flow past a flat plate at a Mach number of 0.5 and a chord Reynolds number of 10,000. The mesh used in the computation is shown in Figure 2a. It contains 2,604 elements, 1,376 points, and 146 boundary points. The computed Mach number contours in the flow field are depicted in figure 2b, where the development of a boundary layer can be clearly observed. Figure 2c shows the comparison of the Blasius velocity profile and the computed velocity profiles as scaled by the Blasius similarity law at different chord length downstream of the leading edge. The computed results indicate that the similarity solution for a flat plate boundary layer is correctly obtained and the solution agrees well with the Blasius solution. Finally, Figure 2d shows a comparison of convergence histories between the explicit and implicit schemes. The explicit scheme results were obtained using three stage Runge-Kutta scheme with implicit residual smoothing and a CFL number of 4.0. The implicit scheme results were obtained using a maximum CFL number of 1,000.

Test case 3. 2D viscous flow past NACA0012 airfoil

The third test case consists of a supersonic low-Reynolds-number flow where the Mach number is 2, the angle of attack is 10 deg, and the Reynolds number is 106. This represents a standard test case, which has received wide attention in the literature and for which experimental data is available. The mesh, containing 29,386 elements, 14,878 points and 370 boundary points after two levels of refinement, is shown in Fig. 3a. The density contours of the computed flow field are depicted in Fig. 3b, where a strong bow shock is observed. The pressure coefficient distribution on the airfoil is shown in Fig.3c. The convergence history is shown in Fig. 3d, where only 60 time steps were required for the residual to drop about 4 orders of magnitude, even with two levels of refinement. The implicit scheme results were obtained using a maximum CFL number of 1,000.

Test case 4. 3D inviscid flow in a channel

The fourth test case is the well known Ni's test case: an inviscid flow in a channel with a 10% thick circular bump on the bottom. Inlet Mach number was 0.675. This is a 3D simulation of a 2D flow. The mesh, which contains 13,891 grid points, 68,097 elements and 4,442 boundary points, is depicted in Fig.4a. Fig.4b displays the computed pressure contours on the surfaces. The Mach number distribution on lower wall is shown in Fig.4c. Fig.4d displays a comparison of convergence histories between the explicit scheme and the implicit scheme with left, right, and symmetric ILU preconditioner, respectively. The explicit scheme results were obtained using three stage Runge-Kutta scheme with implicit residual smoothing and a CFL number of 4. The implicit scheme results were obtained using a CFL number of 100,000. Contrary to the results obtained in reference[10], the left, right and symmetric preconditioners performed equally well.

Test case 5. 3D inviscid flow past a F-117 fighter

The last test case involves a 3D simulation of an inviscid flow past a complete f-117 stealth fighter at a Mach number of 0.8 and an angle of attack of 5 degrees. The mesh, which contains 509,853 elements, 92,854 points, and 12,657 boundary points for the half-span airplane, is shown in Fig.5a. The computed Mach number contours in the flow field are depicted in Fig.5b. For the purpose of comparison, the computation was performed using both explicit and implicit schemes on the Cray-M90 computer at the Cray Research Inc. The solution was converged to engineering accuracy (a decrease of a four order-of-magnitude in the L_2 norm of the density residual). The explicit scheme solution was obtained using three stage Runge-Kutta scheme with implicit residual smoothing and a CFL number of 4. The implicit scheme results were obtained using a maximum CFL number of 10,000. Comparisons of their performance in terms of time steps, CPU time, and storage requirements

are presented in Table 1 below.

Table 1. Time step, CPU, and Storage requirements

	Explicit	Implicit
Time steps	1400	35
CPU	16 hours	3.5 hours
Storage	50 Mwords	165 Mwords

An analysis of these results demonstrates that the implicit scheme runs approximately 4 times faster than its explicit counterpart, while requiring about 3.5 times more memory. It must be remarked that the code includes all extra arrays required for ALE formulation, H-refinement, and remeshing.

7. CONCLUSIONS

An implicit algorithm has been developed for the solution of the compressible Euler and Navier-Stokes equations on unstructured meshes. The treatment of implicit boundary conditions has been found to be critical to the success of an implicit scheme. The numerical results have shown that the present implicit scheme converges to the asymptotic steady state much faster than its explicit counterpart. Overall speed-up factors of up to 5 for the Euler equations and at least one order of magnitude for the Navier-Stokes equations are found in the examples shown.

8. ACKNOWLEDGMENTS

This research was sponsored by the Defense Nuclear Agency. Dr. Michael E. Giltrud served as the technical program monitor. Partial funding for the last two authors was also provided by the Air Force Office of Scientific Research. Dr. Leonidas Sakell served as the technical monitor.

REFERENCES

[1]Luo, H., Baum, J. D., Löhner, R., and Cabello, J., "Adaptive Edge-Based Finite Element Schemes for the Euler and Navier-Stokes Equations on Unstructured meshes," AIAA Paper 93-0336, 1993.

[2]Luo, H., Baum, J. D., and, Löhner, R., "Numerical Solution of the Euler Equations for Complex Aerodynamic Configurations Using an Edge-Based Finite Element Schemes," AIAA Paper 93-2933, 1993.

[3]Peraire, J., Peiro, J., and Morgan, K., "A 3D Finite element Multigrid Solver for the Euler Equations," AIAA Paper 1992-0449, 1992.

[4]Barth, T. J., "Numerical Aspects of Computing Viscous High Reynolds Number Flow on Unstructured Meshes," AIAA Paper 1991-0721, 1991.

[5]Billey, V., Périaux, J., Perrier, P., and Stoufflet, B., "2-D and 3-D Euler Computations with Finite Element Methods in Aerodynamic," International Conference on Hypersonic Problems, Saint-Etienne, Jan. 13-17, 1986.

[6]Venkatakrishnan, V., and Mavriplis, D. J., "Implicit Solvers for Unstructured meshes," AIAA Paper 91-1537, 1991.

[7]Whitaker, D. L., "Solution Algorithms for the Two-Dimensional Euler Equations on Unstructured Meshes," AIAA Paper 90-0697, 1990.

[8]Hassan, O., Morgan, K., and Peraire, J., "An Implicit Finite-Element Method for High Speed Flows," AIAA Paper 90-0402, 1990.

[9]Batina, J. T., "Implicit Flux-Split Euler Schemes for Unsteady Aerodynamic Analysis Involving Unstructured Meshes," AIAA Paper 90-0936, 1990.

[10]Venkatakrishnan, V., "Preconditioned Conjugate Gradient Methods for the Compressible Navier-Stokes Equations," AIAA Paper 90-0586, 1990.

[11]Barth, T. J., "Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes equations," AGARD-R-787, May 1992.

[12]Saad, Y., and Schultz, M. H., "GMRES: a Generalized Minimal Residual Algorithm for Solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comp.*, Vol. 7, No 3 (1988), pp 89-105.

[13]Ajmani, K., Ng, W. F., and Liou, M. S., "Preconditioned Conjugate-Gradient Methods for Low-Speed Flow Calculations," AIAA Paper 1993-0881, 1993.

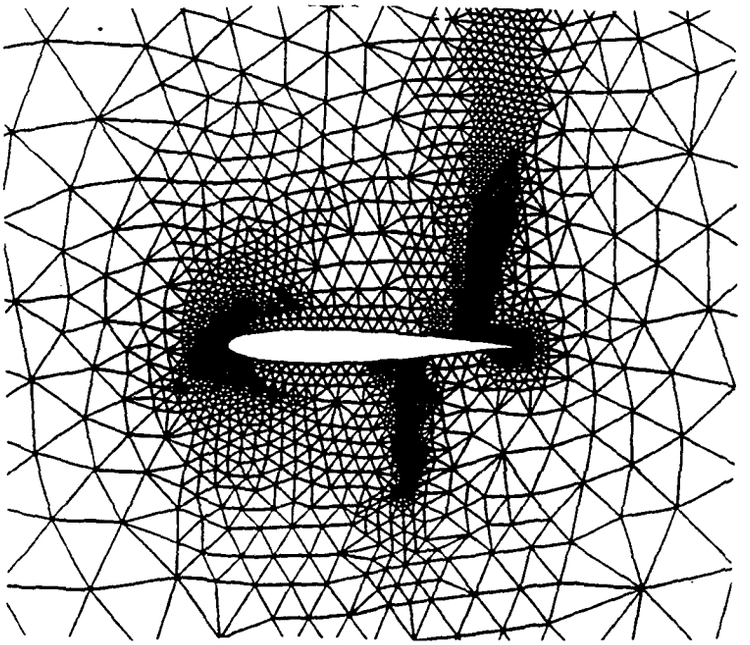


Fig.1a Mesh Used for Computing inviscid Flow past a *NACA0012* airfoil; nelem=6,397, npoin=3,274

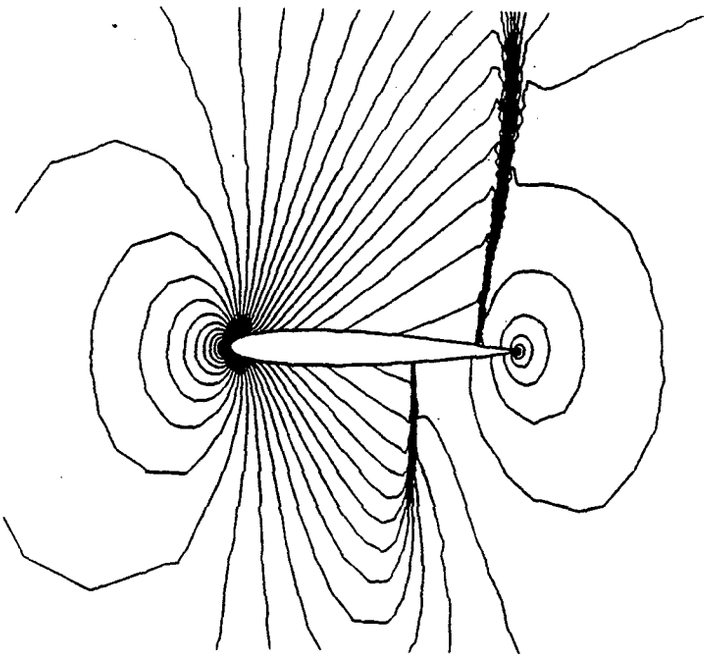


Fig.1b Computed Pressure Contours in the Flow Field; $M_\infty = 0.85, \alpha = 1.0$

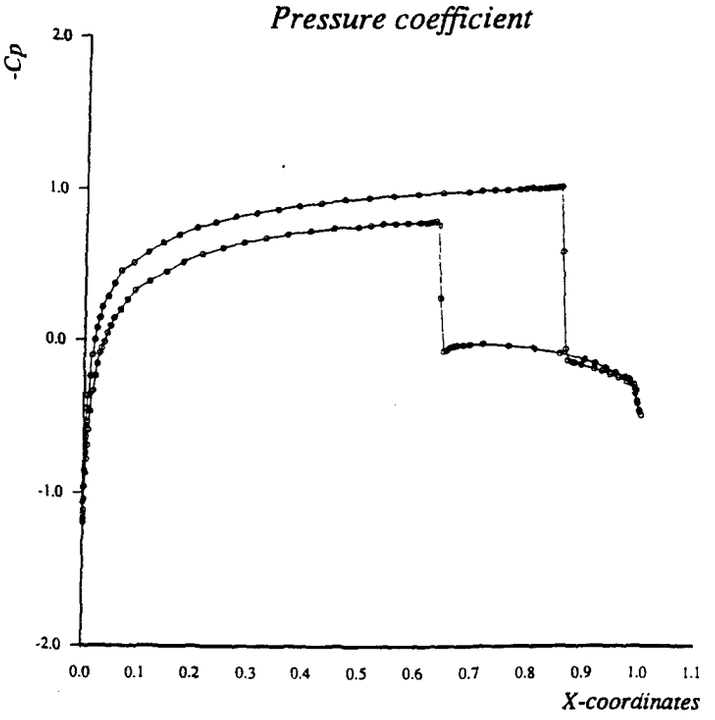


Fig.1c Computed C_p Distribution on the airfoil; $M_\infty = 0.85, \alpha = 1.0$

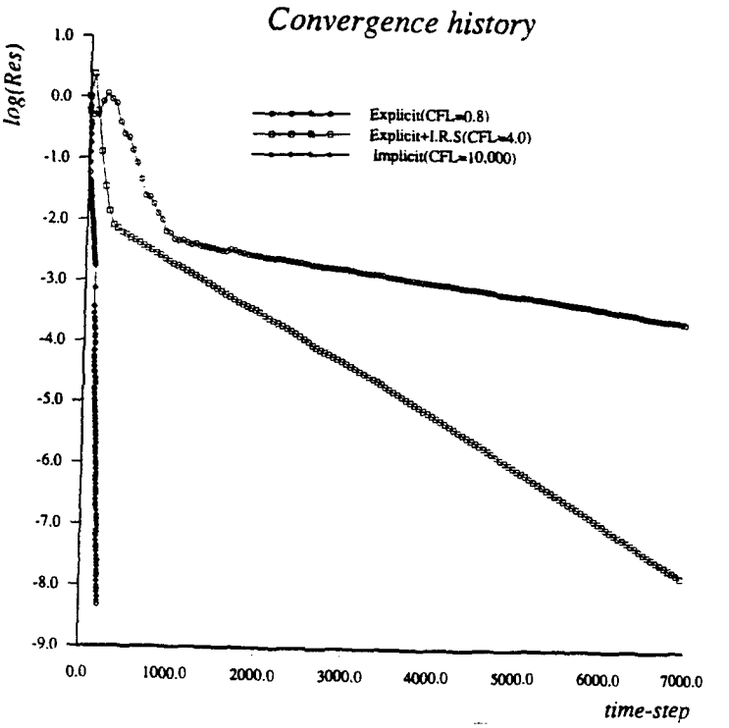


Fig.1d Comparison of convergence history between Explicit and Implicit Schemes

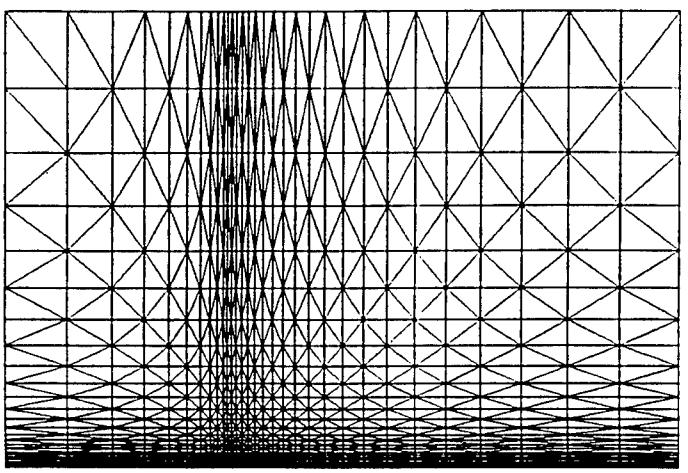


Fig.2a Mesh Used for Computing Viscous Flow past a Flat Plate; nelem=2,604, npoin=1,376

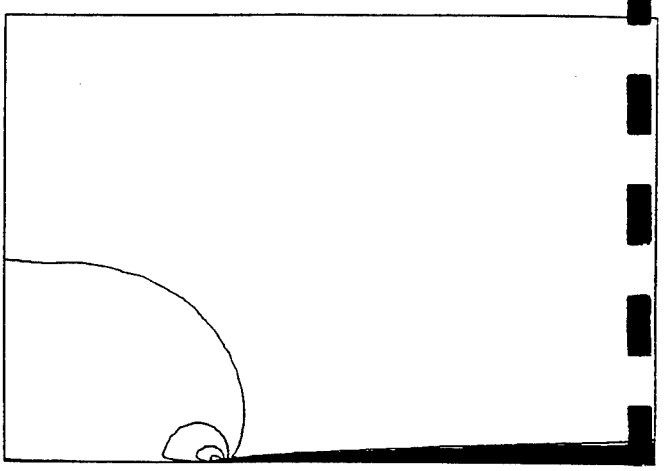


Fig.2b Computed Mach Number Contours past a Flat Plate; $M_\infty = 0.5$, Re=10,000

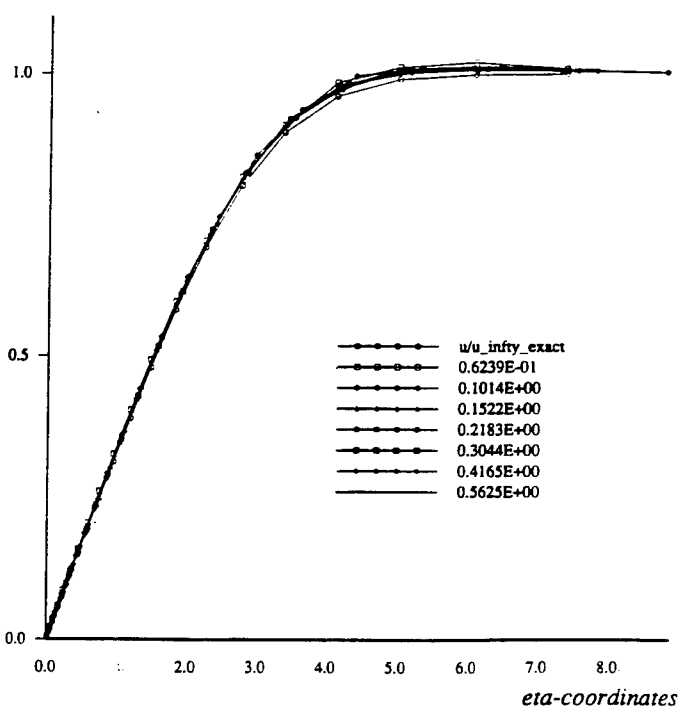


Fig.2c Streamwise Velocity Profiles past a Flat Plate; $M_\infty = 0.5$, Re=10,000

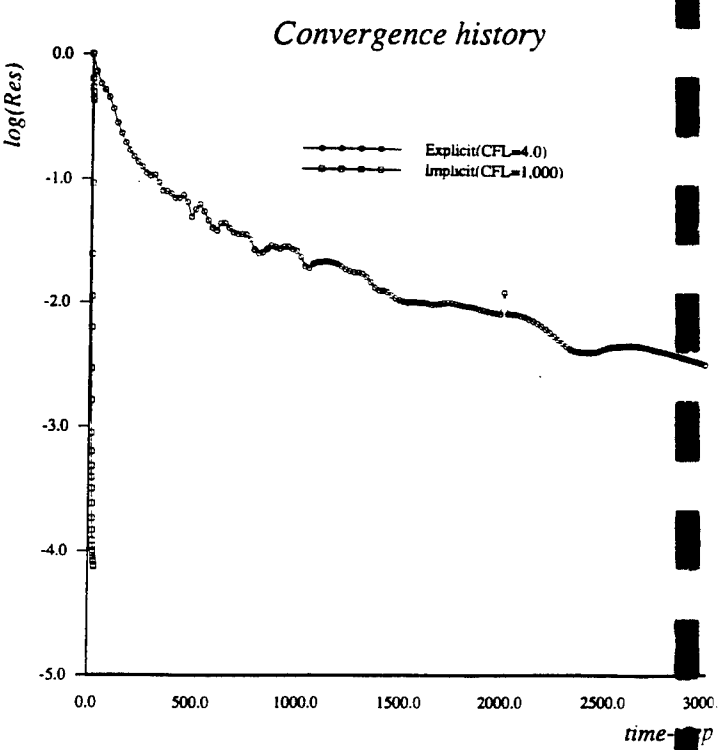


Fig.2d Comparison of convergence history between Explicit and Implicit Schemes

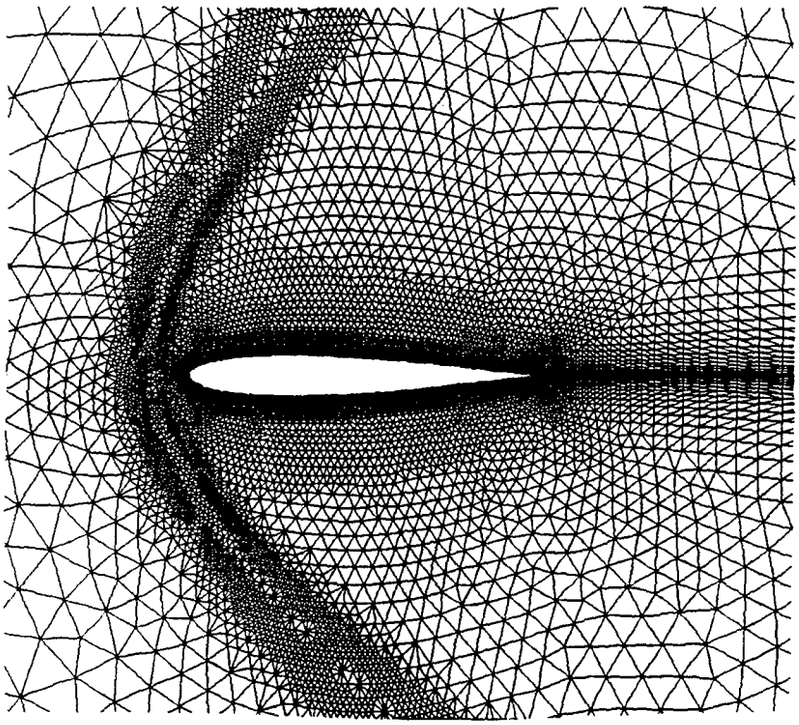


Fig.3a Final Adapted Mesh past a NACA0012 airfoil; melem=29,386, npoin=14,878

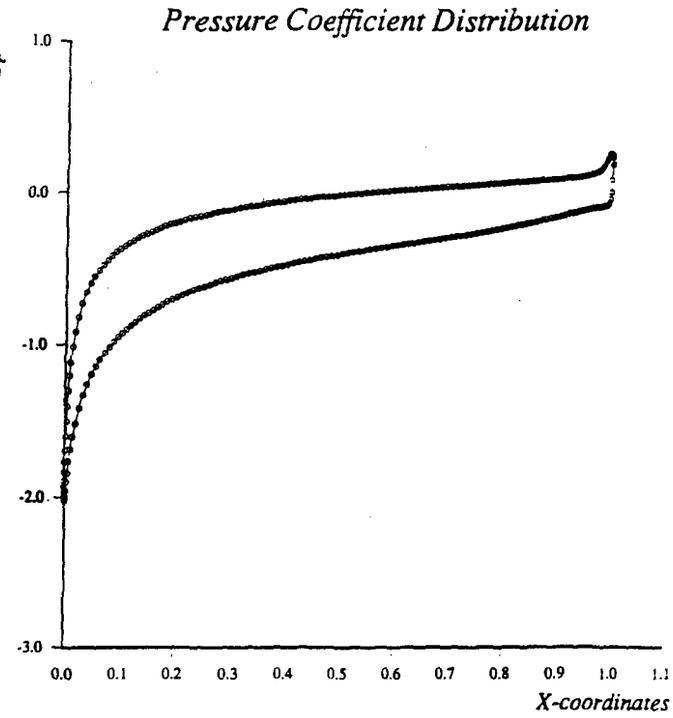


Fig.3c Computed C_p Distribution on the airfoil; $M_\infty = 2, \alpha = 10$,Re=106

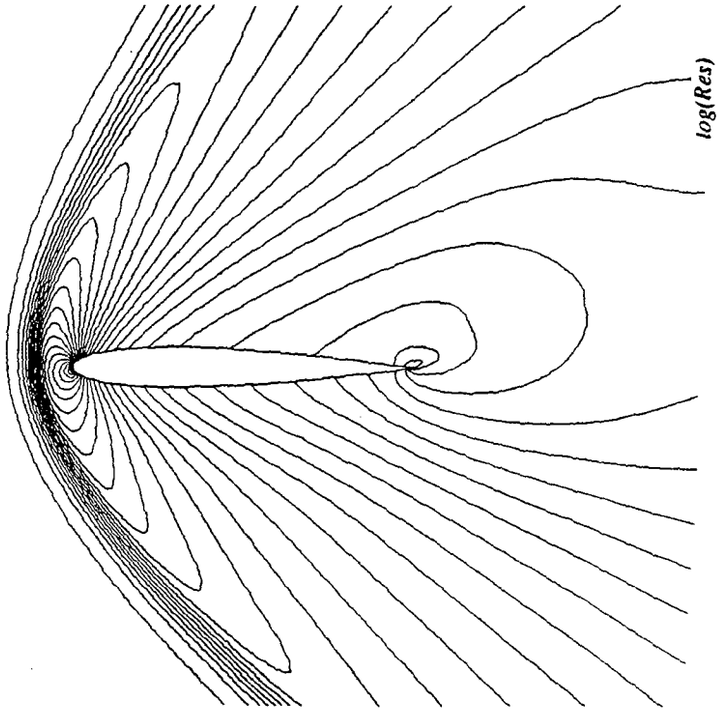


Fig.3b Computed Density Contours past a NACA0012 airfoil; $M_\infty = 2, \alpha = 10$,Re=106

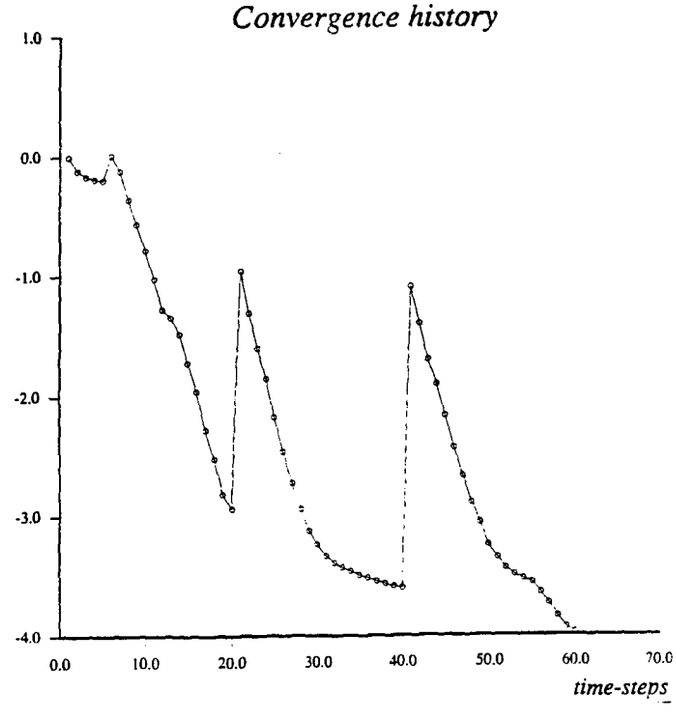


Fig.3d Convergence History for the Implicit Scheme

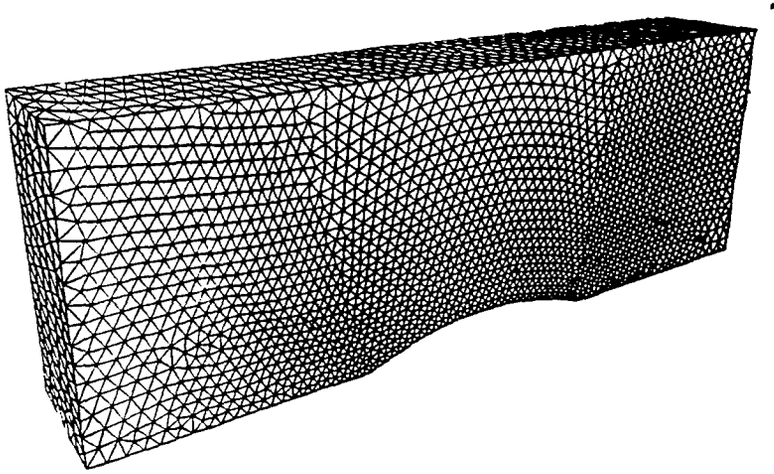


Fig.4a Surface Mesh of Triangles for the Channel
melem=68,097, npoin=13,891, nboun=4,442

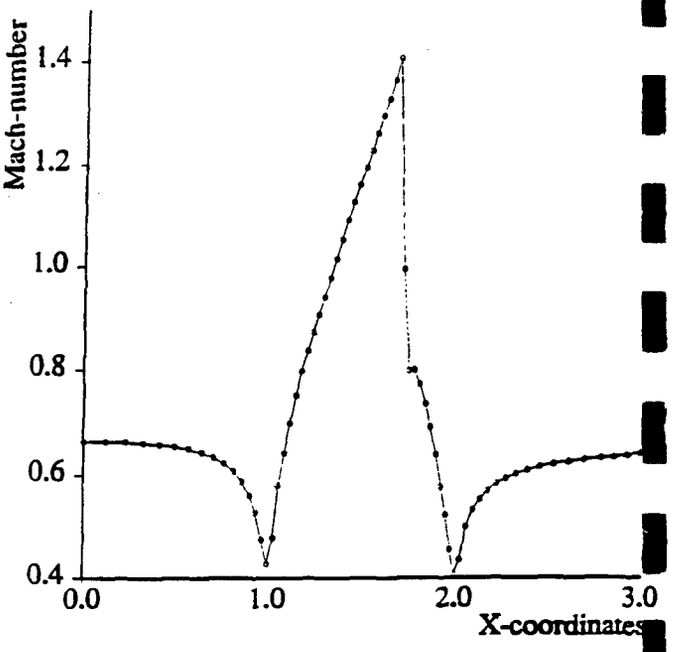


Fig.4c Computed Mach Number Distribution on a lower corner line at $M_\infty = 0.675, \alpha = 0°$

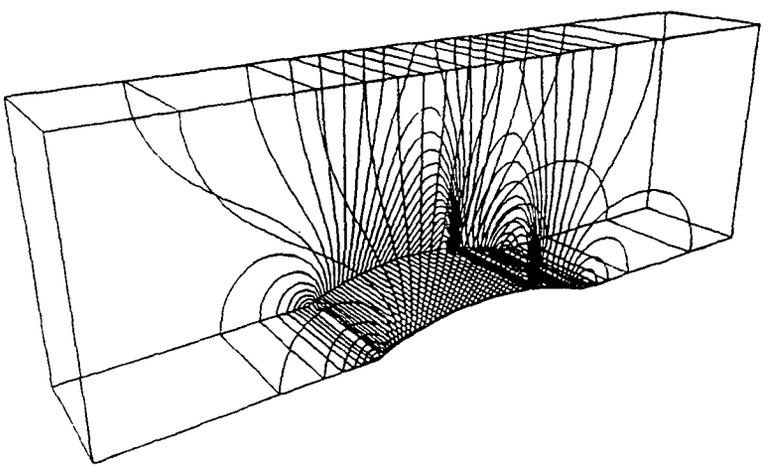


Fig.4b Computed Pressure Contours in the Channel at $M_\infty = 0.675, \alpha = 0°$

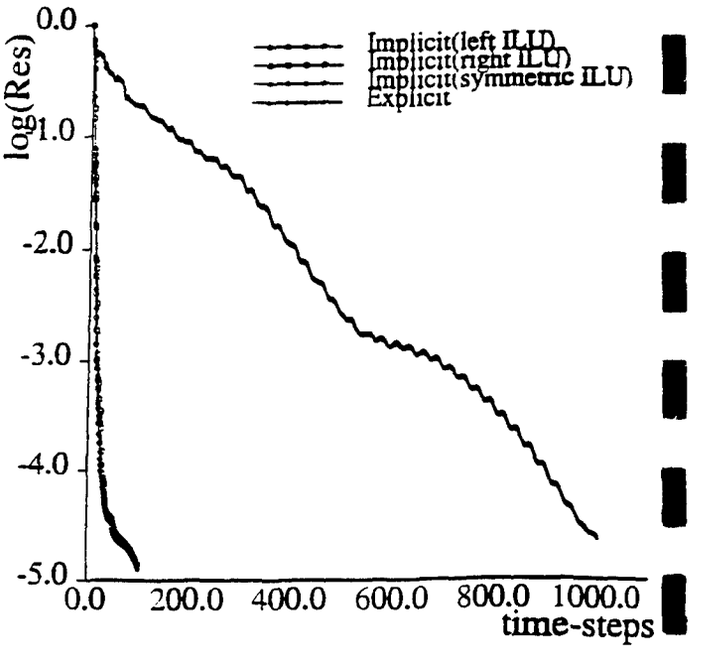


Fig.4d Comparison of Convergence History for Explicit, Left, Right, and Symmetric Precondioners

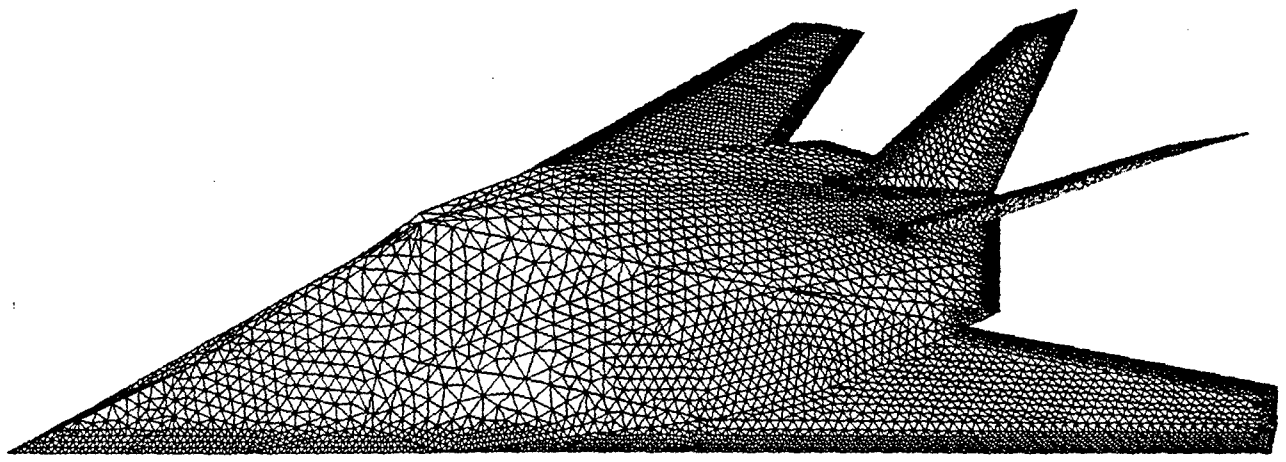


Fig.5a Surface Mesh of Triangles for the F-117 Stealth Fighter
nelem=509,853, npoin=92,854, nboun=12,657

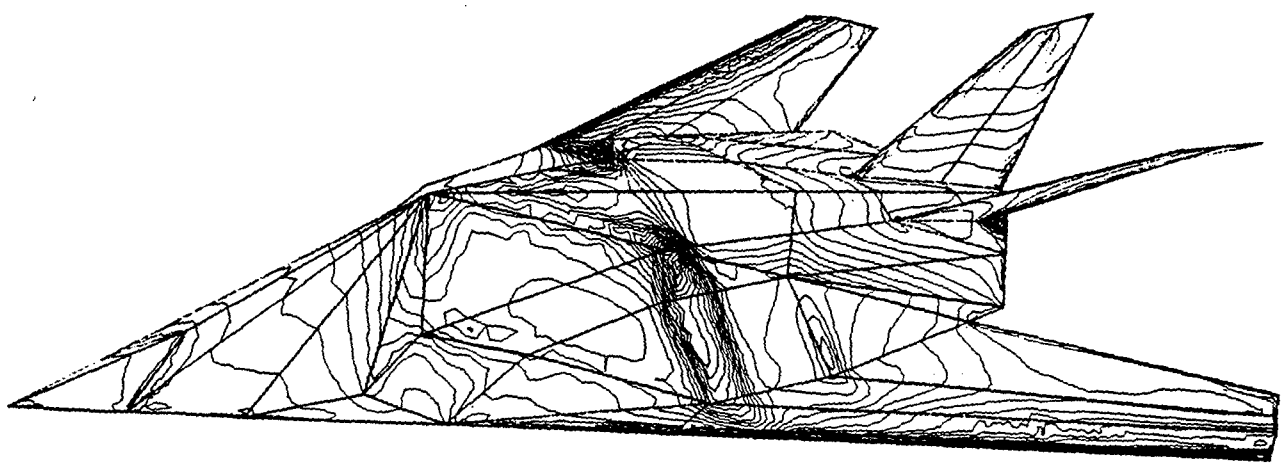


Fig.5(b) Computed Mach Number Contours on the F-117
at $M_\infty = 0.8, \alpha = 5^\circ$

APPENDIX 2: IMPROVED SPATIAL DISCRETIZATION

AIAA 95-0348

An Improved Finite Volume Scheme for Compressible Flows on Unstructured Grids

Hong Luo*, Joseph D. Baum*, and
Rainald Löhner**

* Science Applications International Corporation,
McLean, VA 22102

** George Mason University,
Fairfax, VA 22030

33rd Aerospace Sciences Meeting and Exhibit
January 9-12, 1995 / Reno, NV

AN IMPROVED FINITE VOLUME SCHEME FOR COMPRESSIBLE FLOWS ON UNSTRUCTURED GRIDS

Hong Luo and Joseph D. Baum
Science Applications International Corporation
1710 Goodridge Drive, MS 2-3-1
McLean, VA 22102, USA

and

Rainald Löhner
Institute for Computational Sciences and Informatics
George Mason University, Fairfax, VA 22030, USA

ABSTRACT

This paper describes recent improvements to a node-centered upwind finite volume scheme for the solution of the compressible Euler and Navier-Stokes equations on unstructured meshes. The improvements include a more accurate boundary integration procedure, which is consistent with the finite element approximation, and a new reconstruction scheme based on the consistent mass matrix iteration. Several numerical results are presented to demonstrate the performance of the proposed improvements. The numerical results indicate that the present scheme significantly improves the quality of numerical solutions with very little additional computational cost.

I. INTRODUCTION

The use of unstructured meshes for computational fluid dynamics problems has become widespread due to their ability to discretize arbitrarily complex geometries and due to the ease of adaption in enhancing the solution accuracy and efficiency through the use of adaptive refinement techniques. In recent years, remarkable progress has been made in the development of upwind algorithms for the solution of the Euler and Navier-Stokes equations on unstructured meshes[1-7]. A significant advantage of any upwind discretization is that it is naturally dissipative as compared with central-difference discretizations, and consequently does not require any problem-dependent parameters to adjust.

The present authors have developed an upwind finite element scheme for the solution of the compressible Euler and Navier-Stokes equations[1-2]. It has been found that the finite element scheme gives better results than its finite volume counterpart, although it can be shown that for interior points both schemes yield the same approximation. A detailed examination led us to discover that this discrepancy results from assuming piecewise constant numerical fluxes for the boundary integrals, resulting in a poor approximation for the boundary points. A careful numerical integration formula has to be used in the boundary integrals to get a consistent approximation with the finite element approximation. This consistent boundary integration formula is derived here.

The accuracy of any upwind finite volume scheme is strongly determined by the accuracy of a fundamental process known as reconstruction: e.g. given pointwise values of the solution at the nodes of the mesh, reconstruct the polynomial approximation to the solution in the control volume. It has been demonstrated that the piecewise linear reconstruction methods offer a substantial improvement over the basic first order piecewise constant scheme[8-10]. The piecewise quadratic reconstruction provides a further improvement over its piecewise linear counterpart. However, this is achieved at a very high computational cost[8]. In addition, application of the piecewise quadratic reconstruction scheme to highly stretched meshes remains a problem. Therefore, only piecewise linear reconstruction methods are of interest in this paper. These methods require a best estimate for the solution gradients within each control volume. The most popular linear reconstruction schemes on unstructured meshes are based on either a Green-Gauss formulation or a least-squares principal. Here, we propose a new reconstruction scheme based on the consistent mass-matrix, motivated by the observation that the consistent mass-matrix should be used in the first place, when the solution gradient is computed in a finite element context. Unlike the Green-Gauss or least-squares reconstruction schemes, which rely only upon next-neighbor information, the mass-matrix reconstruction involves information of points beyond nearest neighbors. It is this extra information that yields a more accurate estimation of solution gradients.

Several numerical results for a wide range of flow conditions, from subsonic to supersonic, in both inviscid and viscous flows, are presented to demonstrate the performance of the proposed improved scheme. The numerical results indicate that the mass-matrix reconstruction gives much better results than the Green-Gauss or least-squares schemes and that consistent boundary integration is important to both convergence and accuracy for some test cases.

II. GOVERNING EQUATIONS

The Euler and Navier-Stokes equations governing the unsteady flows can be written in integral form as

$$\frac{\partial}{\partial t}\int_{\Omega} \mathbf{U} d\Omega + \int_{\Gamma} \mathbf{F}\cdot \mathbf{n} d\Gamma = 0, \qquad (1)$$

for a domain Ω with boundary $\Gamma = \partial\Omega$. In this equation, U is the vector of the conservative variables for mass, momentum, and energy. The F represents the inviscid and viscous flux vectors. n denotes the outward normal to the boundary Γ.

III. FINITE VOLUME DISCRETIZATION

The governing equation (1) is discretized using a node-centered finite volume formulation, where flow variables are placed at the nodes of the mesh. The control volume C_i for each node i is taken to be the median dual mesh cells, which are constructed by connecting the centroid of the neighboring cells and the midpoints of the two edges that share the vortex i, as shown in Figure a.

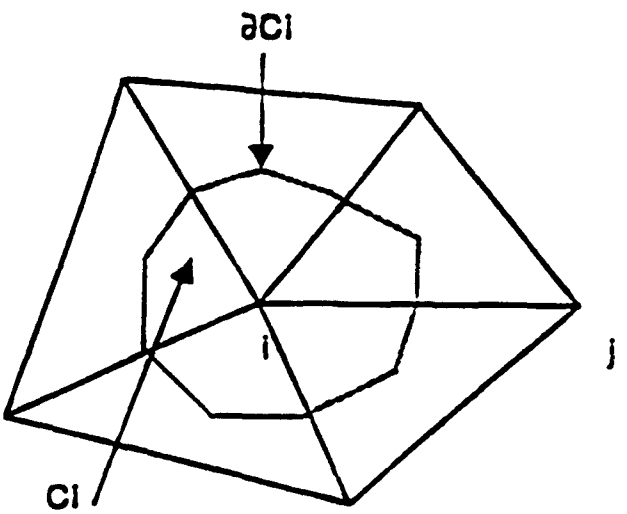


Fig. a Representative Unstructured Grid and Dual Mesh Cell

The finite volume approximation of the governing equation (1), applied to the control volume around node i becomes

$$area(C_i)\frac{d\mathbf{U_i}}{dt} + \int_{\partial C_i} \mathbf{F}\cdot \mathbf{n} d\Gamma = 0, \qquad (2)$$

where ∂C_i is the boundary of the control volume C_i.

The flux integral in equation (2) is evaluated by summing all the contributions over the cell interfaces between the node i and its neighboring node j, $\partial C_{ij} (= \partial C_i \cap \partial C_j)$. Equation (2) can then be rewritten in a compact form as

$$\mathbf{M_i}\frac{d\mathbf{U_i}}{dt} = -\mathbf{R_i}, \qquad (3)$$

where $\mathbf{M_i}$ is the volume of the dual mesh cell (equivalent to the lumped mass matrix in the finite element), and $\mathbf{R_i}$ is the right hand side residual,

$$\mathbf{R_i} = -\sum_j \int_{\partial C_{ij}} \mathbf{F}\cdot \mathbf{n} d\Gamma - \int_{\partial C_i \cap \Gamma} \mathbf{F}\cdot \mathbf{n} d\Gamma. \qquad (4)$$

The numerical fluxes on the interface ∂C_{ij} are approximated at the mid-point of edge ij, and the integral along the interface can then be evaluated as

$$\int_{\partial C_{ij}} \mathbf{F}\cdot \mathbf{n} d\Gamma = \mathbf{F}_{ij}\cdot \mathbf{N}_{ij}, \qquad (5)$$

where $\mathbf{N}_{ij} = \int_{\partial C_{ij}} \mathbf{n} d\Gamma$ denotes the normal to the interface ∂C_{ij}.

It is clear that the right hand side is formed by two loops; one is over the edges of the mesh, the other over the boundary faces. It can be readily shown that if a linear shape function for the fluxes is used in the finite element method, i.e.,

$$\mathbf{F} = \sum \mathbf{F}_i N_i \qquad (6)$$

where N_i is the standard linear finite element shape function associated with the node i, and a linear interpolation used for the fluxes in the finite volume method, i.e.,

$$\mathbf{F}_{ij} = \frac{1}{2}(\mathbf{F}_i + \mathbf{F}_j), \qquad (7)$$

both approximations would produce the same results for an interior point (i.e. the same right hand side). However, this is not necessarily true for a boundary point, where the results depend on how to one computes the boundary integral in Eq. (4). If the boundary integral is computed using a piecewise constant approximation for the numerical fluxes, i.e.,

$$\int_{\partial C_i \cap \Gamma} \mathbf{F}\cdot \mathbf{n} d\Gamma = \sum \mathbf{F}_i \cdot \mathbf{n}_{ij}\frac{L_{ij}}{2}, \qquad (8)$$

where the summation is over all the boundary faces attaching the node i, $\mathbf{n}_{ij}$ represents the unit vector normal to the boundary face ij, and L_{ij} denotes the length of boundary face ij, they will give different right hand side. The natural question to be asked at this point is then which one gives the better approximation. To illustrate this point, we compared the velocity field for a potential flow in a channel using both the finite volume approach and the finite element approach. Note that the finite element approach is equivalent to the common area-weighted averaging approach. Figures b and c display the velocity contours obtained by finite volume and finite element approximations, respectively. One can clearly see that the finite volume approach gives an erroneous solution on the boundaries, while the finite element approach produces a correct solution. Assuming that both approaches produce the same results for a boundary point, it is a direct but lengthy process to show that the following integration formula should be used in the boundary integral for the finite volume approach:

$$\int_{\partial C_i \cap \Gamma} \mathbf{F}\cdot \mathbf{n} d\Gamma = \sum \frac{5\mathbf{F}_i + \mathbf{F}_j}{6}\cdot \mathbf{n}_{ij}\frac{L_{ij}}{2}, \qquad (9)$$

in 2-D, and

$$\int_{\partial C_i \cap \Gamma} \mathbf{F}\cdot\mathbf{n}d\Gamma = \sum \frac{6\mathbf{F}_i + \mathbf{F}_j + \mathbf{F}_k}{8}\cdot\mathbf{n}_{ijk}\frac{L_{ijk}}{3}, \quad (10)$$

in 3-D. In general, the difference between using the piecewise constant approximation and the above averaged formulas in the boundary integrals for the Euler equations has little effect on both solution convergence and accuracy. But for some cases, the difference can be so significant that the consistent integration formulas (9) and (10) are recommended for use.

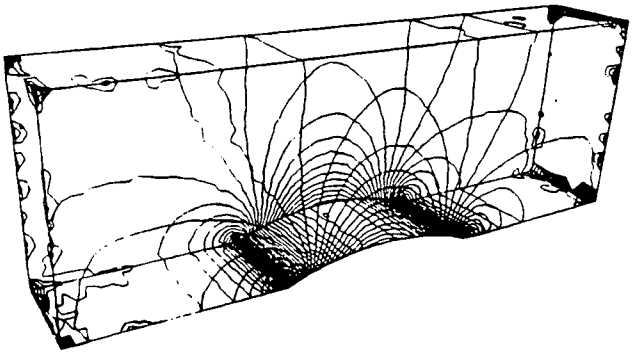


Fig. b Potential velocity contours obtained using finite volume approach

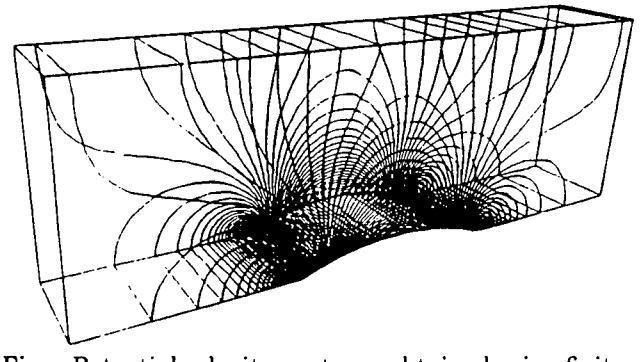


Fig. c Potential velocity contours obtained using finite element approach

As mentioned earlier, if the numerical fluxes at the interface in equation (5) are simply evaluated as an arithmetical average of the normal fluxes, the resulting finite volume scheme, equivalent to the classic Galerkin finite element scheme, allows for for the appearance of checkerboarding modes, and thus suffers from numerical instabilities, unless some type of numerical dissipation in the form of artificial viscosity is introduced. To construct a stable scheme for the Euler equations, any of the Riemann solvers can be formulated by adopting different forms for the numerical fluxes at the interface. In the present work, a stable scheme is obtained by using one of the most popular approximate Riemann solvers, namely the flux-difference splitting of Roe[11]:

$$\mathbf{F}_{ij}\cdot\mathbf{N}_{ij} = \frac{1}{2}(\mathbf{F}_i + \mathbf{F}_j)\cdot\mathbf{N}_{ij} - \frac{1}{2}\mid \mathbf{A}(\mathbf{U}_i, \mathbf{U}_j, \mathbf{N}_{ij}) \mid (\mathbf{U}_j - \mathbf{U}_i)\ , \quad (11)$$

where $\mid \mathbf{A} \mid$ denotes the standard Roe matrix evaluated in the direction $\mathbf{N}_{ij}$. In equation (11), the solution is assumed to be piecewise constant per control volume, and the resulting upwind scheme is only first order accurate in space. To achieve higher order accuracy, the solution is assumed to be piecewise linear in the control volume, and the numerical fluxes at the interface is evaluated using upwind-biased interpolations of the solution U via the MUSCL approach[12]. This leads to the numerical fluxes

$$\mathbf{F}_{ij}\cdot\mathbf{N}_{ij} = \frac{1}{2}(\mathbf{F}_i^+ + \mathbf{F}_j^-)\cdot\mathbf{N}_{ij} - \frac{1}{2}\mid \mathbf{A}(\mathbf{U}_i^+, \mathbf{U}_j^-, \mathbf{N}_{ij}) \mid (\mathbf{U}_j^- - \mathbf{U}_i^+) \quad (12)$$

where

$$\mathbf{F}_i^+ = \mathbf{F}(\mathbf{U}_i^+), \qquad \mathbf{F}_j^- = \mathbf{F}(\mathbf{U}_j^-)\ . \quad (13)$$

The upwind-biased interpolations for $\mathbf{U}_i^+$ and $\mathbf{U}_j^-$ are defined by

$$\mathbf{U}_i^+ = \mathbf{U}_i + \frac{1}{2}\phi\mathbf{l}_{ij}\cdot\nabla\mathbf{U}_i\ , \quad (14)$$

and

$$\mathbf{U}_j^- = \mathbf{U}_j - \frac{1}{2}\phi\mathbf{l}_{ij}\cdot\nabla\mathbf{U}_j\ , \quad (15)$$

where $\mathbf{l}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ is the length vector of this edge and ϕ is the flux limiter.

IV. RECONSTRUCTION SCHEMES

As seen above, accurate reconstruction is the key ingredient in extending a first order upwind scheme to higher order spatial accuracy on unstructured meshes. The reconstruction algorithm consists of finding a polynomial representation to the solution in each control volume, given pointwise values of a solution at nodes of the mesh. For the piecewise linear reconstruction, where a linear polynomial approximation to the solution is generated in each control volume, the computation of solution gradients at nodes of the mesh is simply required. In this section, three methods of computing solution gradients will be addressed and discussed.

a. Green-Gauss reconstruction

The most commonly used and the simplest reconstruction scheme is the Green-Gauss gradient reconstruction. This gradient calculation is obtained by using the control volume approach and applying the Green-Gauss's theorem,

$$\mathbf{M}_l(\nabla\mathrm{u})_i = \int_{\partial C_i} \mathrm{u}\cdot\mathbf{n}d\Gamma. \quad (16)$$

As mentioned in the previous section, this reconstruction scheme is equivalent to the area-weighted averaging approximation in the finite element, if the consistent integration formulas (9) and (10) are used in the boundary integral.

b. Least-Squares reconstruction

Least squares reconstruction provides an alternative method for computing solution gradients. The computation of solution gradients are performed in the form of a minimization problem. The complete details of this reconstruction procedure can be find in reference 8. However, the procedure will be summarized here for completeness. Consider a node i and assume that the solution veries linearly along an edge ij. Then, the change in node values of the solution along this edge can be computed by

$$(\nabla \mathrm{u})_i \cdot (\mathrm{r_j} - \mathrm{r_i}) = \mathrm{u_j} - \mathrm{u_i}. \tag{17}$$

Similar equations could be written for all edges connected to node i, subject to an arbitrary weighting factor w_i. This yields the following non-square matrix

$$\begin{pmatrix} w_1\Delta x_1 & w_1\Delta y_1 \\ \vdots & \vdots \\ w_n\Delta x_n & w_1\Delta y_n \end{pmatrix} \begin{pmatrix} \mathrm{u_x} \\ \mathrm{u_y} \end{pmatrix} = \begin{pmatrix} w_i(\mathrm{u_1} - \mathrm{u_i}) \\ \vdots \\ w_n(\mathrm{u_n} - \mathrm{u_i}) \end{pmatrix}, \tag{18}$$

which can be solved using the least squares method. The algorithm can be implemented using the edge-based data structure at a cost comparable to that of the Green-Gauss reconstruction.

This formulation provides a freedom in the choice of weighting coefficients w_i. These weighting coefficients can be selected as a function of the geometry and/or solution. Classical approximations in one dimension can be recovered by choosing geometrical weights of the form $w_i = 1.0/\mid \mathrm{r_i} - \mathrm{r_j} \mid^t$ for values of $t = 0, 1, 2$. The numerical computations shown in the next section were performed using $t = 1$.

c. Consistent mass reconstruction

The common feature of the previous two reconstruction schemes is that the estimate of the solution gradients relies only on next-neighbor information. It is apparent that information at points beyond nearest neighbor must be involved to get a more accurate estimation of solution gradients. This can be achieved using the consistent mass matrix instead of the lumped mass matrix in the Green-Gauss' reconstruction scheme, i.e.,

$$\mathbf{M_c} \nabla \mathrm{u} = \mathbf{R}, \tag{19}$$

which, in fact, should have been used in the first place. The consistent mass matrix was replaced by diagonal, lumped mass matrix only for computational expediency. As $\mathbf{M_c}$ possesses an excellent condition number, equation (19) is never solved directly, but iteratively. This is done using an iterative procedure of the form:

$$\mathbf{M_l}(\nabla \mathrm{u}^k - \nabla \mathrm{u}^{k-1}) = \mathbf{R} - \mathbf{M_c} \nabla \mathrm{u}^{k-1}, 1 \leq k \leq \mathrm{niter} \tag{20}$$

where $\nabla \mathrm{u}^k$ denotes the kth iterate. Typically, three passes are required to converge. The solution of equation (20) can also be obtained using an edge-based data structure. The computational cost for solving this equation is very small compared to the overall cost for any upwind type schemes.

V. NUMERICAL RESULTS

All computations used an explicit three-stage Runge-Kutta time-stepping scheme with local time stepping and implicit residual smoothing for advancing the solution to steady state. The solutions were obtained by converging the residual to computer machine zero. Wherever possible, the solutions were obtained using a second order scheme without any limiters, in an effort to ensure that the solution accuracy is affected only by the reconstruction schemes, not by the limiters.

Test case 1. Supersonic vortex flow

The problem under consideration is an inviscid supersonic vortex flow. This test case was selected to compare the order of accuracy and discretization error associated with Green-Gauss, least-squares, and consistent mass reconstruction schemes, since an exact, closed form, analytical solution exists for such flow. Since this is a shock free compressible flow, the solution is obtained using a second order scheme without any limiters. Thus, this test case provides a good opportunity to compare the accuracy of each reconstruction schemes, without any influence of limiters. By comparing the error in the discrete solutions on a successively refined sequence of meshes, quantitative measurements of both order of accuracy and absolute error are possible. For each reconstruction scheme, solutions are sought on a set of three telescoping grids with 31X31, 61X11, and 121X21 nodes. The Mach number at the inner radius r_i is specified at 2.25 and the outer radius r_o at 1.384 r_i. Figure 1a shows the three sets of regular meshes used in the simulation. Figure 1b provides the details of the spatial accuracy of each reconstruction scheme for this numerical experiment. The results indicate that the consistent-mass matrix reconstruction gives better results than the other two in terms of both order of accuracy and absolute error.

Test case 2. Subsonic flow in a channel

The second test case presents an inviscid subsonic flow in a channel with a 10% thick circular bump on the bottom. The Mach number at the inflow is 0.1. The solution is obtained using a second order scheme without any limiters. The mesh, which contains 839 grid points, 1,559 elements and 117 boundary points, is depicted in Fig. 2a. Figure 2b displays the computed velocity contours in the flow field obtained by the three different reconstruction schemes. Figure 2c shows the velocity distributions on the lower wall obtained by the three reconstruction schemes. For comparison purposes, we add the potential solution velocity distribution. Entropy distribution results are displayed in Fig. 2d. The results indicate that the consistent-mass reconstruction generates the least numerical entropy, and produces a virtually identical solution to the potential solution. The significant improvement of the consistent mass reconstruction over

the two other reconstruction schemes can clearly be observed in Fig. 2c.

Test Case 3. NACA0012 Airfoil

The problem under consideration is transonic flow around a NACA0012 airfoil with a freestream Mach number of 0.8, and an angle of attack of 1.25 degrees - a classical test problem for Euler solvers. The solution is obtained using van Albada limiter. The mesh containing 6,891 elements, 3,537 points, and 183 boundary points is displayed in Fig. 3a. Figures. 3b to 3d show the computed pressure contours in the flow fileds obtained by the three reconstructions schemes, respectively. The pressure coefficient distributions on the airfoil obtained by three reconstruction schemes are shown in Figs. 3f-h. Figure 3e shows a comparison of entropy distribution on the airfoil for the three reconstruction schemes. It is observed that the consistent mass matrix reconstruction gives not only oscillation-free strong shock on the upper surface, but also a sharper weak shock on the lower surface.

Test Case 4. Viscous Flow Past a Flat Plate

This test case involves a laminar flow past a flat plate at a Mach number of 0.2 and a chord Reynolds number of 10,000. The computation is performed using a second order scheme without any limiters. The mesh used in the computation is shown in Fig. 4a, and contains 2,604 elements, 1,376 points, and 146 boundary points. Figure 4b shows a comparison of the Blasius velocity profiles and the computed velocity profiles as scaled by the Blasius similarity law at x/L=0.2585. The numerical results indicate that all three reconstruction schemes give virtually identical x-direction velocity distribution. Nevertheless, the consistent mass matrix reconstruction scheme yields the best y-direction velocity distribution.

VI. CONCLUSIONS

Recent improvements to a node-centered upwind finite volume scheme are presented for the solution of the Euler and Navier-Stokes equations on unstructured meshes. A more accurate boundary integration procedure, consistent with the finite element approximation, is formulated. A new reconstruction scheme based on the consistent mass matrix iteration is developed. A variety of flow problems are computed to demonstrate the improvements of the proposed scheme. The numerical results indicate that the consistent mass matrix reconstruction produces the most accurate gradient estimates, and thus significantly improves the quality of numerical solutions with very little additional computational cost.

Acknowledgments

This research was sponsored by the Defense Nuclear Agency. Dr. Michael E. Giltrud served as the technical program monitor. Partial funding for the third author was also provided by the Air Force Office of Scientific Research. Dr. Leonidas Sakell served as the technical monitor.

References

[1] H. Luo, J. D. Baum, R. Löhner and J. Cabello, "Adaptive Edge-Based Finite Element Schemes for the Euler and Navier-Stokes Equations on Unstructured meshes," AIAA Paper 93-0336, Jan. 1993.

[2] H. Luo, J. D. Baum, and R. Löhner, "Edge-Based Finite Element Scheme for the Euler Equations," *AIAA Journal*, Vol. 32, No. 6, 1994, pp.1183-1190.

[3] J. Peraire, J. Peiro and K. Morgan, "A 3D Finite element Multigrid Solver for the Euler Equations," AIAA Paper 92-0449, Jan. 1992.

[4] V. Billey, J. Périaux, P. Perrier, B. Stoufflet, "2-D and 3-D Euler Computations with Finite Element Methods in Aerodynamic," *International Conference on Hypersonic Problems*, Saint-Etienne, Jan. 13-17 (1986).

[5] Timothy J. Barth and Dennis C. Jespersen, "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper 89-0366, Jan. 1989.

[6] David L. Whitaker, "Solution Algorithms for the Two-Dimensional Euler Equations on Unstructured Meshes," AIAA Paper 90-0697, Jan. 1990.

[7] J. T. Batina, "Three-Dimensional Flux-Split Euler Schemes Involving Unstructured Meshes," AIAA Paper 90-1649, Jan. 1990.

[8] T. J. Barth, "Recent Developments in High Order K-Exact Reconstruction on Unstructured Meshes," AIAA Paper 93-0668 Jan. 1993.

[9] M. Aftosmis, D. Gaitonde, and T. Sean Tavares, "On the Accuracy, Stability and Monotonicity of Various Reconstruction Algorithms for Unstructured meshes," AIAA Paper 94-0415, Jan. 1994.

[10] C. R. Mitchell, "Improved Reconstruction Schemes for the Navier-Stokes Equations on Unstructured Meshes," AIAA Paper 94-0642, Jan. 1994.

[11] P. L. Roe, "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes," *Journal of Computational Physics*, 43 (1981), pp. 357-372.

[12] Van Leer, "Towards the Ultimate Conservative Difference Scheme, II. Monotonicity and Conservation Combined in a Second Order Scheme," *Journal of Computational Physics*, 14 (1974), pp. 361-370.

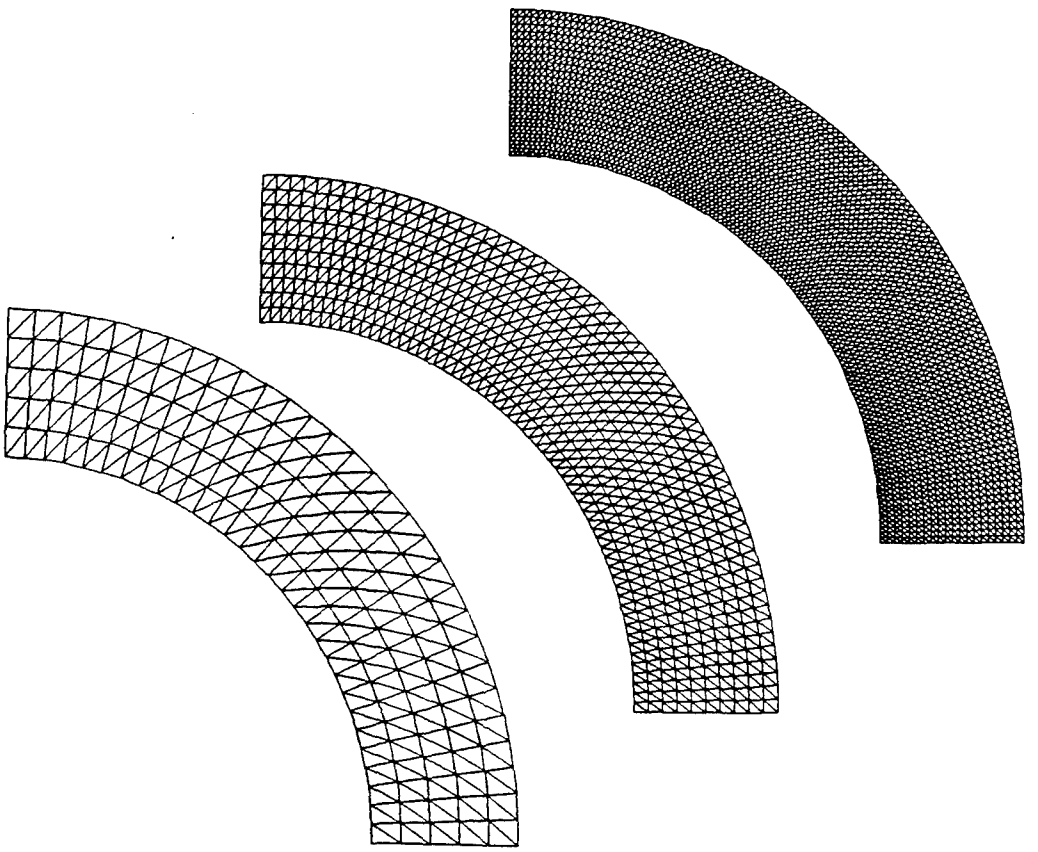


Fig.1a Sequences of meshes used for supersonic vertex flow

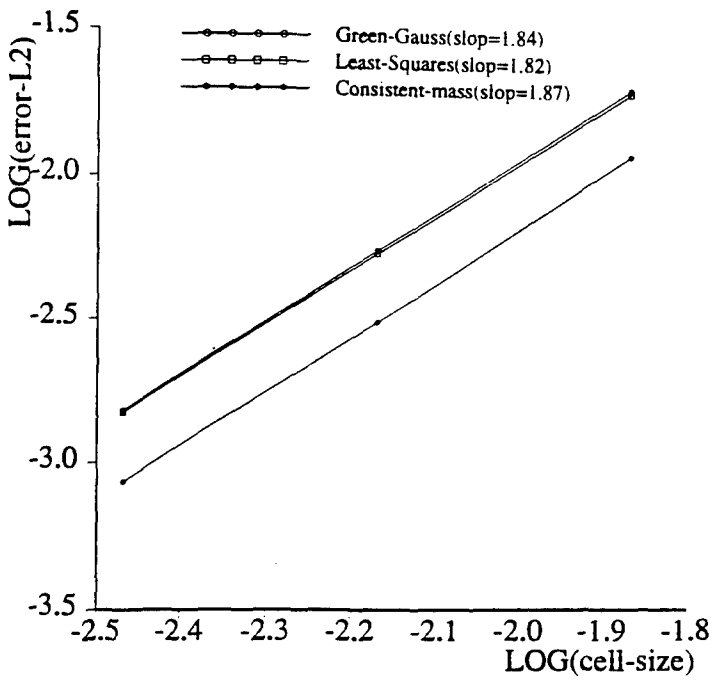


Fig.1b Accuracy summary for different reconstruction schemes

APPENDIX 3: MESH MOVING TECHNIQUES

COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING, Vol. 12, 599–608 (1996)

IMPROVED ALE MESH VELOCITIES FOR MOVING BODIES

RAINALD LÖHNER AND CHI YANG
GMU/CSI, George Mason University, Fairfax, VA 22030-4444, USA

SUMMARY

A Laplacian smoothing of the mesh velocities with variable diffusivity based on the distance from moving bodies is introduced. This variable diffusivity enforces a more uniform mesh velocity in the region close to the moving bodies. Given that in most applications these are regions where small elements are located, the new procedure decreases element distortion considerably, reducing the need for local or global remeshing, and in some cases avoiding it altogether.

KEY WORDS finite elements; moving grids; moving bodies; mesh velocity; ALE

1. INTRODUCTION

For any arbitrary Lagrangean–Eulerian (ALE) unstructured-grid field solver that considers bodies or surfaces in relative motion to one another, a recurring question has been how to specify the mesh velocity of the field points.[1-10] In mathematical terms: given the velocity $\mathbf{w}$ on the moving surfaces:

$$\mathbf{w}|_{\Gamma_0} = \mathbf{w}_0, \tag{1}$$

and, at a certain distance from these moving surfaces, as well as all the remaining surfaces, a vanishing mesh velocity

$$\mathbf{w}|_{\Gamma_1} = 0, \tag{2}$$

find the spatial distribution of $\mathbf{w}$ such that element distortion is minimized. If this mesh velocity distribution is not smooth, distorted elements will appear quickly, forcing many local or global remeshings, with the ensuing loss of accuracy and increase in CPU requirements. Three families of methods have been used to specify the mesh velocity:

(a) Analytic user-prescribed functions,
(b) Smoothing of coordinates, and
(c) Smoothing of velocities.

In the first case, the mesh velocity is prescribed to be an analytic function of the distance from the surface. Efficient distance-from-body search algorithms are nowadays common,[11-14] albeit scalar. Given the distance from moving surfaces δ, and the point on the surface closest to it $\mathbf{x}|\Gamma$, the mesh velocity at any field point is given by

$$\mathbf{w} = \mathbf{w}(\mathbf{x}|_\Gamma) f(\delta). \tag{3}$$

CCC 1069–8299/96/100599–10

Received 24 October 1995
Revised 11 March 1996

The function $f(\delta)$ assumes the value of unity for $\delta = 0$, and decays to zero as δ increases. This makes the procedure somewhat restrictive for general use, particularly if several moving bodies are present in the flowfield. On the other hand, the procedure is extremely fast if the initial distance δ can be employed for all times.[6,8]

In the second case, the edges of the unstructured grid are treated as springs that are relaxed in time to achieve equilibrium. In this way, a uniform element distribution is maintained. Starting from the prescribed boundary velocities, a new set of boundary coordinates is obtained at the new time step:

$$\mathbf{x}^{n+1}|_\Gamma = \mathbf{x}^n|_\Gamma + \Delta t\, \mathbf{w}|_\Gamma. \tag{4}$$

Based on these new values for the coordinates of the boundary points, the mesh is smoothed. Although more sophisticated mesh smoothing techniques have been proposed,[10] by far the most common way to smooth this new mesh is via spring analogy relaxation.[3,7] The force exerted by each spring (edge) is a function of its length and acts along its direction. Therefore, the sum of the forces exerted by all springs surrounding a point can be written as

$$\mathbf{f}_i = \sum_{j=1}^{ns_i} c(|\mathbf{x}_j - \mathbf{x}_i|)(\mathbf{x}_j - \mathbf{x}_i), \tag{5}$$

where c denotes the spring function, $\mathbf{x}_i$ the coordinates of the point, and the sum extends over all the points surrounding the point. At the surface of the computational domain, no movement of points is allowed, i.e. $\Delta \mathbf{x} = 0$. The new values for the coordinates are obtained iteratively via a relaxation or conjugate gradient scheme.[3,7] Once the new coordinates have been evaluated, the mesh velocity is computed from

$$\mathbf{w} = \frac{1}{\Delta t}(\mathbf{x}^{n+1} - \mathbf{x}^n). \tag{6}$$

Most of the potential problems that may occur for this type of mesh velocity smoothing are due to initial grids that have not been smoothed. For such cases, the velocity of the moving boundaries is superposed to a ficticious mesh smoothing velocity which may be quite large during the initial stages of a run. Moreover, for spring analogy smoothers there is no guarantee that negative elements will not appear.

In the third case, the mesh velocity is smoothed directly, based on the exterior boundary conditions given by (1), (2). The aim, as stated before, is to obtain a mesh velocity field w in such a way that element distortion is minimized. Consider for the moment the 1-D situation sketched in Figure 1. At the left end of the domain, the mesh velocity is prescribed to be w_0. At

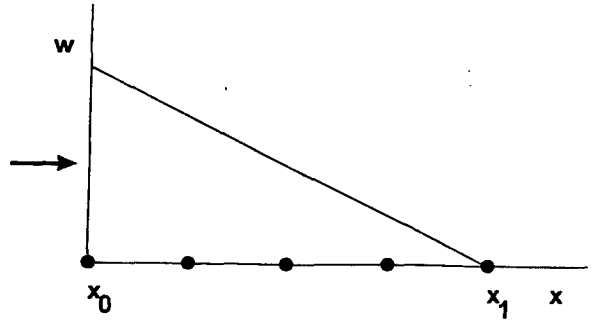


Figure 1. Mesh velocity smoothing in 1-D

the right end, the mesh velocity vanishes. If the mesh velocity decreases *linearly*, i.e.

$$\frac{\partial w}{\partial x} = g_v, \tag{7}$$

then the elements will maintain their initial size ratios. This is because for any two elements the change in size δh during one time step is given by

$$\delta h = (w_2 - w_1)\Delta t = \Delta w\, \Delta t. \tag{8}$$

This implies that for the size ratio of any two elements i, j we obtain

$$\left.\frac{h_i}{h_j}\right|^{n+1} = \frac{h_i|^n + \Delta w_i \Delta t}{h_j|^n + \Delta w_j \Delta t} = \frac{h_i|^n + g_v h_i|^n \Delta t}{h_j|^n + g_v h_j|^n \Delta t} = \left.\frac{h_i}{h_j}\right|^n, \tag{7}$$

i.e. all elements in the regions where mesh velocity is present will be deformed in roughly the same way. Solutions with constant gradients are reminiscent of Laplacian operators, and indeed, for the general case, the mesh velocity may be obtained by solving

$$\nabla \mathbf{k} \nabla \mathbf{w} = 0, \tag{10}$$

with the Dirichlet boundary conditions given by (1), (2). This system is discretized using finite element procedures. The resulting system of equations can be solved in a variety of ways, e.g. via relaxation as

$$C^{ii}\, \Delta \mathbf{w}^i = -\Delta t\, K^{ij}(\mathbf{w}^i - \mathbf{w}^j), \tag{11}$$

where

$$C^{ii} = \sum_{i \neq j} |K^{ij}|, \tag{12}$$

and the optimal Δt-sequence is given by

$$\Delta t^i = \frac{1}{1 + \cos\left[\dfrac{\pi(i-1)}{n}\right]}, \qquad i = 1, n. \tag{13}$$

If the diffusion coefficient appearing in (10) is set to $k=1$, a true Laplacian velocity smoothing is obtained. This yields the most 'uniform deformation' of elements, and therefore minimizes the number of remeshings or remappings required. Alternatively, for element-based codes, one may approximate the Laplacian coefficients K^{ij} in (11) by

$$\nabla^2 \mathbf{w} \approx -(\mathbf{M}_l - \mathbf{M}_c)\mathbf{w}, \tag{14}$$

where $\mathbf{M}_l$, $\mathbf{M}_c$ denote, respectively, the lumped and consistent mass matrices.This approximation is considerably faster for element-based codes (for edge-based codes there is no difference in speed between the true Laplacian and this expression), but it is equivalent to a diffusion coefficient $k = h^2$. This implies that the gradient of the mesh velocity field will be larger for smaller elements. These will therefore distort at a faster rate than the larger elements. Obviously, for uniform grids this is not a problem, but in many cases the smallest elements are close to the surfaces that move, prompting many remeshings.

Based on the previous arguments, one may also consider a diffusion coefficient of the form $k = h^{-p}$, $p > 0$. In this case, the gradient of the mesh velocity field will be larger for the larger

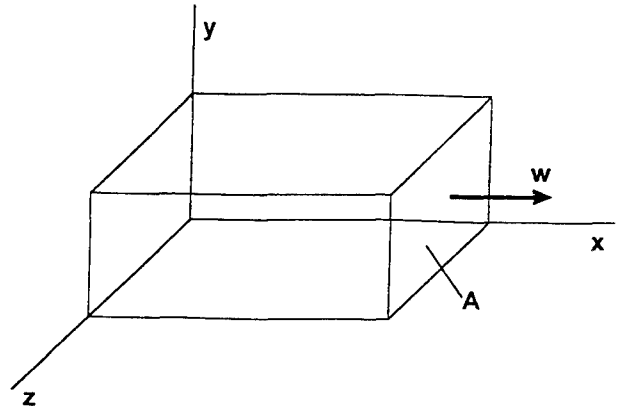


Figure 2. 1-D movement of face A

elements. The larger elements will therefore distort at a faster rate than the smaller ones – a desirable feature for many applications.

To see more clearly the difference between mesh velocity and coordinate smoothers, consider the simple box shown in Figure 2. Suppose that face A is being moved in the x-direction. For the case of coordinate smoothing, there will, in all likelihood, appear mesh velocities in the y- and z-directions. This is because, as the mesh moves, the smoothing technique will result in displacements of points in the y- and z-directions, and hence velocities in the y- and z-directions. On the other hand, for the case of mesh velocity smoothing, only displacements in the x-direction will appear. This is because the Dirichlet boundary conditions given by (1), (2) do not allow any mesh velocity other than in the x-direction to appear. We consider this an advantage of mesh velocity smoothers, and have therefore pursued them from the outset.

2. VARIABLE DIFFUSIVITY LAPLACIAN SMOOTHING

In most practical applications, the relevant flow phenomena and associated gradients of density, velocity and pressure are on or close to the bodies immersed in the fluid. Hence, the smallest elements are typically encountered close to the bodies. A straightforward Laplacian smoothing of the mesh velocities will tend to distort the elements in these critical regions. Thus, the small elements in the most critical regions tend to be the most deformed, leading to a loss in accuracy and possible reinterpolation errors due to the high rate of remeshings required. In an attempt to mitigate this shortcoming, we propose a diffusion coefficient k that is based on the distance δ from the moving bodies. In general, k should be a function of δ as sketched in Figure 3. For small δ, k should be large, leading to a small gradient of $\mathbf{w}$, i.e. nearly constant mesh velocity close to the moving bodies. For large δ, k should tend to unity in order to ensure the most uniform deformation of the (larger) elements that are away from the bodies.

2.1. Distance evaluation

The calculation of the distance δ can be carried out in a variety of ways. Scalar, optimal search procedures have been employed within grid generation and turbulence modelling,[11-14] but for purposes of parallelization we prefer the Laplacian-based distance evaluation detailed

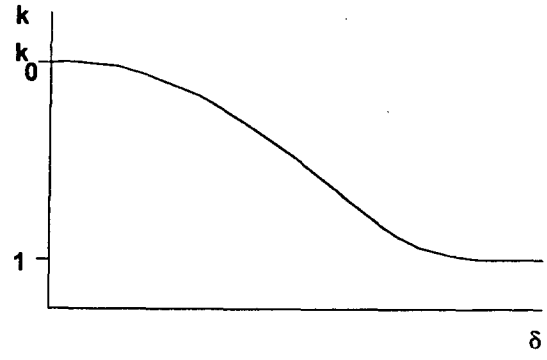


Figure 3. General shape for desired diffusivity k

here. Consider the 1-D Poisson problem:

$$\delta_{,xx} = -s; \qquad \delta(0) = 0; \qquad \delta_{,x}|_{x_1} = 0. \tag{15}$$

The exact solution is given by

$$\delta = sx\left(x_1 - \frac{x}{2}\right), \tag{16}$$

implying

$$x_1 = \sqrt{\frac{2\delta_1}{s}}; \qquad \delta_{,x}|_0 = \sqrt{2s\delta_1}. \tag{17}$$

This means that in order to obtain a unit gradient at $x=0$, one should choose $s=1/2\delta_1$, which in turn leads to $x_1 = 2\delta_1$. Another way to interpret the results is that the 'rigidization' distance x_1 is related to the maximum value of $\delta = \delta_1$ and the source strength s.

This simplified analysis is not valid for 2-D and 3-D solutions of the general Poisson problem

$$\nabla^2\delta = -s; \qquad \delta|_{\Gamma_0} = 0; \qquad \delta_{,n}|_{\Gamma_1} = 0, \tag{18}$$

where for radial symmetry the solutions contain $\ln(r)$ and $1/r$ terms. On the other hand, we do not require the exact distance from moving bodies, but only a distance function that will give the proper behaviour for k. We therefore use (18) to determine the distance function δ. The Poisson problem is solved using finite element procedures and an iterative procedure similar to that given by (11)–(13) above. This fits naturally into existing CFD codes, where edge-based Laplacian operator modules exist for artificial or viscous dissipation terms. The Neumann condition in (18) is enforced by not allowing δ to exceed a certain value. After each iterative pass during the solution of (18), we impose

$$\delta \leqslant \delta_1, \tag{19}$$

which in effect produces the desired Neumann boundary condition at Γ.

2.2. *Diffusivity as a function of distance*

As stated before, for small δ, k should be large, leading to a very small gradient of $\mathbf{w}$, i.e. nearly constant mesh velocity close to the moving bodies. For large δ, k should tend to unity in

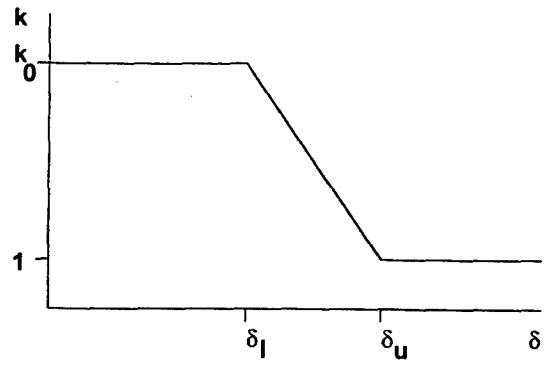


Figure 4. Diffusivity as a function of distance

order to ensure the most uniform deformation of the (larger) elements that are away from the bodies. For the diffusion k, we use the following constant-linear-constant function (see Figure 4):

$$k = k_0 + (1 - k_0)\max\left(0, \min\left(1, \frac{\delta - \delta_l}{\delta_u - \delta_l}\right)\right). \tag{20}$$

The choice of the cut-off distances is, in principle, arbitrary. We have found $\delta_l = x_1/4$, $\delta_u = x_1/2$ to be a good choice.

3. EXAMPLES

The procedure outlined above has been used extensively within an unstructured-grid, edge-based ALE CFD code.[9] The usefulness of changing k according to the distance as given by (20) is demonstrated on a moving wing as well as a hypersonic store release case computed recently.

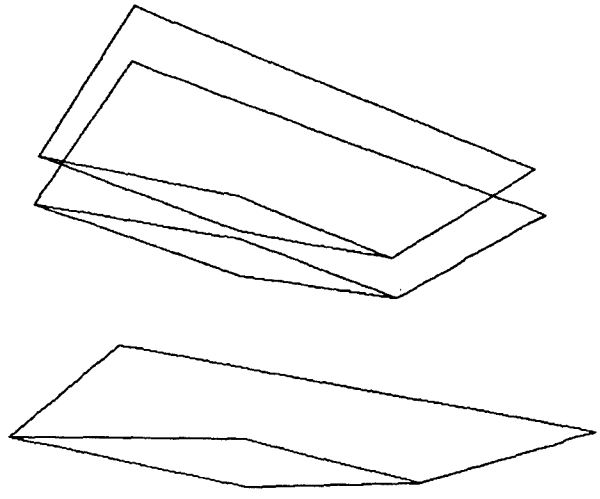


Figure 5. Outline of wind after 0, 30 and 100 time steps

3.1. Wing

A moving wing is first considered using both Laplacian ($k_0 = 1$, $x_1 = 0$) and modified Laplacian ($k_0 = 100$, $x_1 = 2$) velocity smoothing. Figure 5 shows the outline of the wing after 0, 30 and 100 time steps. The surface grids and mesh velocities obtained using the two methods after 30 time steps are shown in Figures 6 and 7. As a result of the larger gradient of the mesh velocity field, the meshes in the vicinity of the wing start becoming distorted in the case of Laplacian velocity smoothing, and the first negative element appears after 34 time steps. On the other hand, for the modified Laplacian velocity smoothing the meshes in the vicinity of the wing remain undistorted, and no remeshing is required even after 100 time

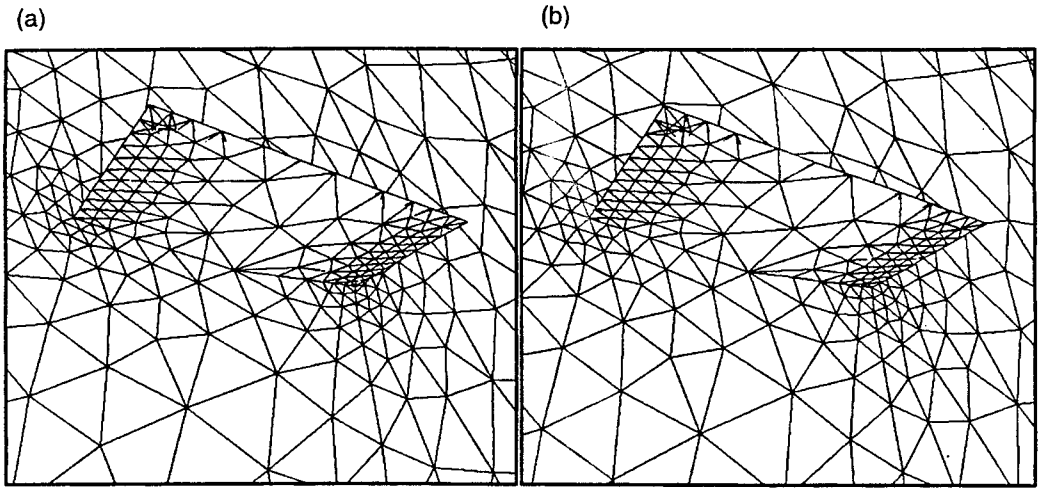


Figure 6. Surface mesh after 30 time steps: (a) Laplacian velocity smoothing; (b) modified Laplacian velocity smoothing

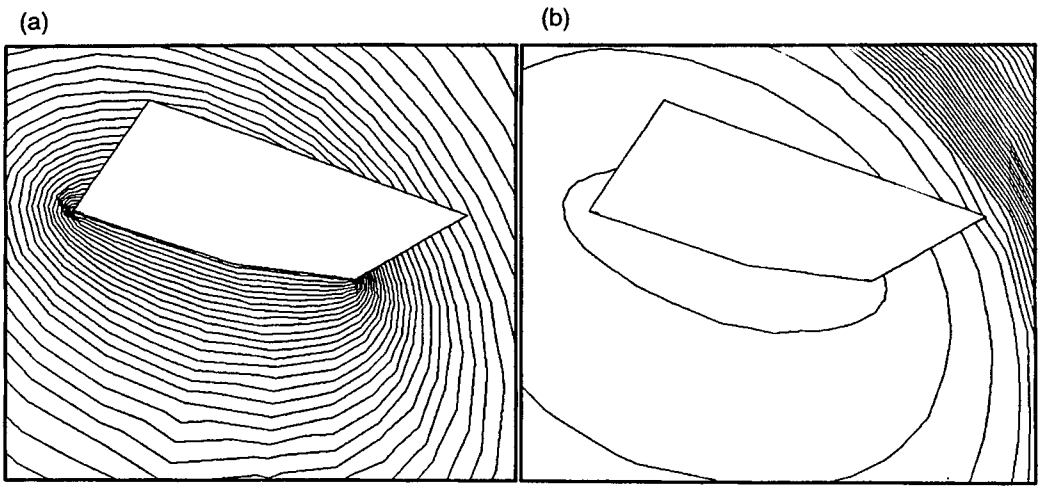


Figure 7. Surface mesh velocity after 30 time steps: (a) Laplacian velocity smoothing ($\Delta \mid v \mid = 0{\cdot}05$); (b) modified Laplacian velocity smoothing ($\Delta \mid v \mid = 0{\cdot}05$)

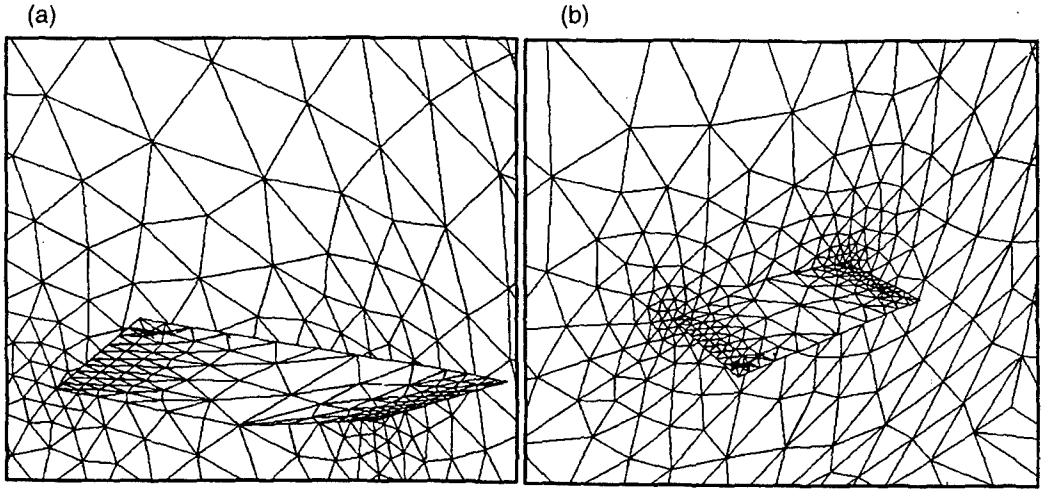


Figure 8. Surface mesh after 100 time steps (modified Laplacian velocity smoothing)

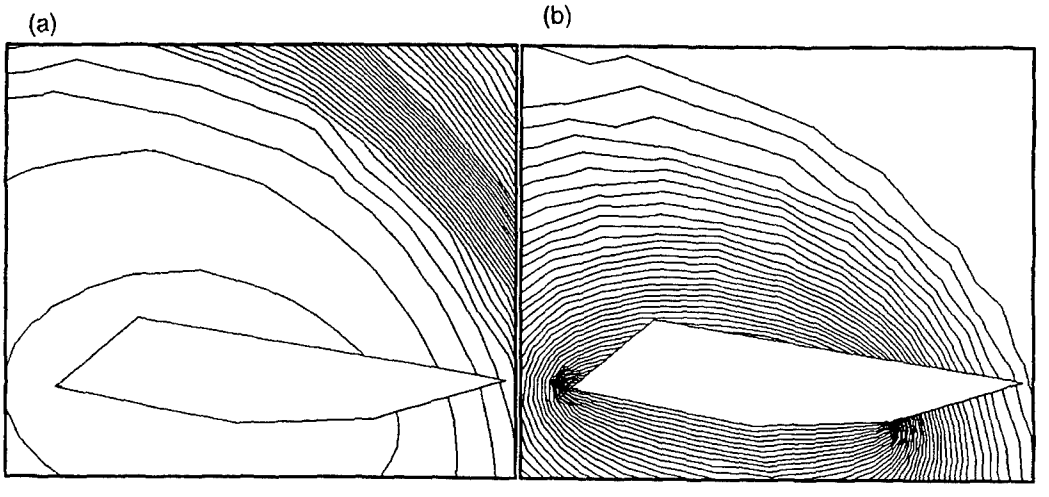


Figure 9. (a) Surface mesh velocity after 100 time steps ($\Delta \mid v \mid = 0{\cdot}05$); (b) distance function ($\Delta\delta = 0{\cdot}05$); (modified Laplacian velocity smoothing)

steps. The surface grid and mesh velocity, as well as the distance function after 100 time steps are shown in Figures 8 and 9, respectively.

3.2. Hypersonic store release

As a second, more realistic case, we consider a hypersonic store release. From a given state, we followed the solution for 100 time steps, setting in the first case $k_0 = 1$, $x_1 = 0$, and in the second case $k_0 = 50$, $x_1 = 0{\cdot}08$. The surface grids and mesh velocities obtained after 100 time steps are displayed in Figures 10 and 11. As one can see, the new variable k mesh velocity smoothing leads to a much less deformed grid close to the moving missile. For this case, the

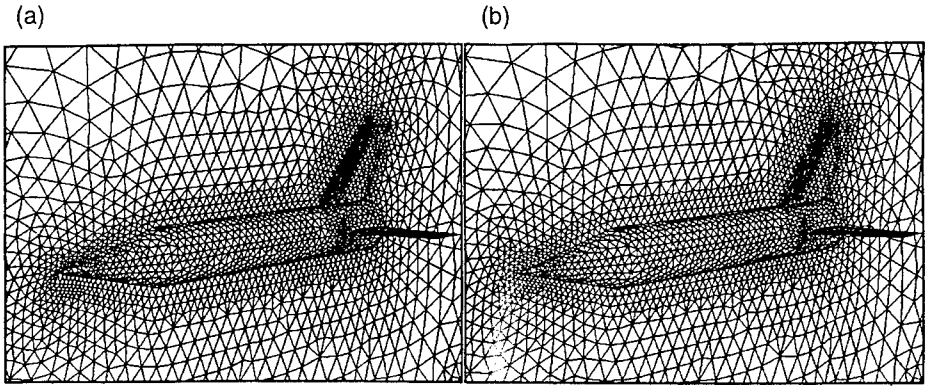


Figure 10. Surface mesh after 100 time steps: (a) Laplacian velocity smoothing; (b) modified Laplacian velocity smoothing

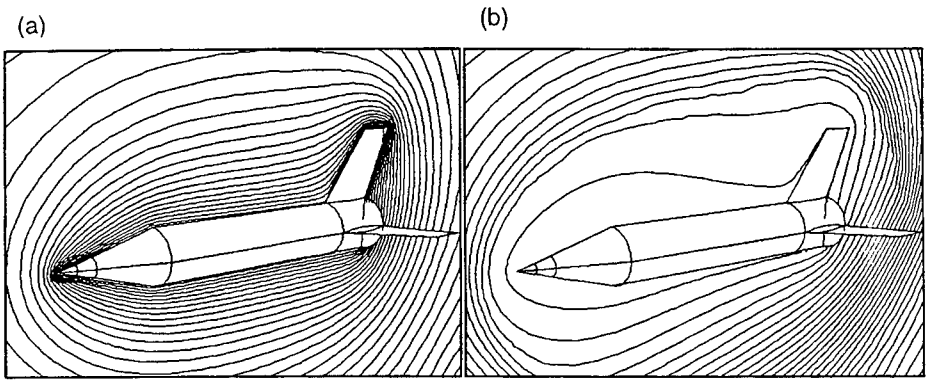


Figure 11. Surface mesh velocity after 100 time steps: (a) Laplacian velocity smoothing; (b) modified Laplacian velocity smoothing

number of local remeshings required dropped by a factor of 1 : 4, leading to considerable CPU savings in a multiprocessor environment.

4. CONCLUSIONS

A Laplacian smoothing of the mesh velocities with variable diffusivity based on the distance from moving bodies has been found effective for unstructured-grid ALE solvers. The variable diffusivity enforces a more uniform mesh velocity in the region close to the moving bodies. Given that in most applications these are regions where small elements are located, the new procedure decreases element distortion considerably, reducing the need for local or global remeshing, and in some cases avoiding it altogether.

As the mesh movement is linked to a time-stepping algorithm for the fluid part, and the body movement occurs at a slow pace compared to the other wavespeeds in the coupled fluid/solid system, normally no more than five steps are required to smooth the velocity field sufficiently, i.e. $3 \leqslant n \leqslant 5$ in (11). The overhead incurred by this type of smoothing is very small compared to the overall costs for any ALE-type methodology for Euler or Navier–Stokes flow solvers.

ACKNOWLEDGEMENTS

This work was partially funded by AFOSR under contract F496209410119, with Dr. Leonidas Sakell as the technical monitor.

REFERENCES

1. L. Formaggia, J. Peraire and K. Morgan, 'Simulation of a store separation using the finite element method', *Appl. Math. Model.*, **12**, 175–181 (1988).
2. R. Löhner, 'An adaptive finite element solver for transient problems with moving bodies', *Comput. Struct.*, **30**, 303–317 (1988).
3. J. T. Batina, 'Unsteady Euler airfoil solutions using unstructured dynamic meshes', *AIAA J.*, **28**(8), 1381–1388 (1990).
4. R. Löhner, 'Three-dimensional fluid-structure interaction using a finite element solver and adaptive remeshing', *Comput. Syst. Eng.*, **1**(2–4), 257–272 (1990).
5. J. D. Baum and R. Löhner, 'Numerical simulation of pilot/seat ejection from an F-16', *AIAA-93-0783*, (1993).
6. A. H. Boschitsch and T. R. Quackenbush, 'High accuracy computations of fluid-structure interaction in transonic cascades', *AIAA-93-0485*, (1993).
7. R. D. Rausch, J. T. Batina and H. T. Y. Yang, 'Three-dimensional time-marching aeroelastic analyses using an unstructured-grid Euler method', *AIAA J.*, **31**(9), 1626–1633 (1993).
8. G. A. Davis and O. O. Bendiksen, 'Unsteady transonic two-dimensional Euler solutions using finite elements', *AIAA J.*, **31**, 1051–1059 (1993).
9. J. D. Baum, H. Luo and R. Löhner, 'A new ALE adaptive unstructured methodology for the simulation of moving bodies', *AIAA-94-0414*, (1994).
10. V. Venkatakrishnan and D. J. Mavriplis, 'Implicit method for the computation of unsteady flows on unstructured grids', *AIAA-95-1705-CP*, (1995).
11. P. Rostand, 'Algebraic turbulence models for the computation of 2-D high speed flows using unstructured grids', *ICASE Rep. 88-63*, (1988).
12. R. Löhner, 'Some useful data structures for the generation of unstructured grids', *Commun. Appl. Numer. Methods*, **4**, 123–135 (1988).
13. J. Bonet and J. Peraire, 'An alternate digital tree algorithm for geometric searching and intersection problems', *Int. J. Numer. Methods Eng.*, **31**, 1–17 (1991).
14. D. Martin and R. Löhner, 'An implicit linelet-based solver for incompressible flows', *AIAA-92-0668*, (1992).

APPENDIX 4: FAST INTERPOLATION SCHEMES

JOURNAL OF COMPUTATIONAL PHYSICS 118, 380–387 (1995)

Robust, Vectorized Search Algorithms for Interpolation on Unstructured Grids

RAINALD LÖHNER

GMU/CSI, The George Mason University, Fairfax, Virginia 22030-4444

Received October 19, 1993; revised November 21, 1994

Several search algorithms for the interpolation of data associated with unstructured grids are reviewed and compared. Particular emphasis is placed on the pitfalls these algorithms may experience for grids commonly encountered and on ways to improve their performance. It is shown how the most CPU-intensive portions of the search process may be vectorized. A technique for the proper interpolation of volumetric regions separated by thin surfaces is included. Timings for several problems show that speedups in excess of 1:5 can be obtained if due care is used when designing interpolation algorithms. © 1995 Academic Press, Inc.

1. INTRODUCTION

The need to interpolate quickly the fields of unknowns from one mesh to another is common to many areas of computational mechanics and computational physics. The following classes of problems require fast interpolation algorithms:

(a) *Simulations where the grid changes as the solution proceeds.* Examples of this kind are adaptive remeshing for steady-state and transient simulations [1–3], as well as remeshing for problems where grid distortion due to movement becomes too severe [4, 5].

(b) *Loose coupling of different codes for multi-disciplinary applications.* In this case, if any of the codes in question are allowed to perform adaptive mesh refinement, the worst case scenario requires a new interpolation problem at every timestep.

(c) *Interpolation of discrete data for the initialization or continuous update of boundary conditions.* Common examples are meteorological simulations, as well as climatological and geotechnical data for seepage and surface flooding problems.

(d) *Visualization.* This large class of problems makes extensive use of interpolation algorithms, in particular for the comparison of different data sets on similar problems.

The main reason that prompted us to revisit the search and interpolation problem was the second class of applications. We are currently developing a series of loosely coupled multidisciplinary codes. We have found that for these classes of problems, interpolation can take a non-negligible portion of total CPU-time, especially for large applications running on multiprocessor vector-computers.

In the following, we will concentrate on the fast interpolation between different unstructured grids that are composed of the same type of elements. In particular, we will consider linear triangles and tetrahedra. The ideas developed are general and can be applied to any type of element and grid. On the other hand, other types of grids (e.g., cartesian structured grids) will lend themselves to specialized algorithms that may be more efficient and easier to implement.

The remainder of the paper is organized as follows. Section 2 describes the basic algorithm used to decide if a point of the unknown grid is inside an element of the known grid. Sections 3–5 consider the fastest possible algorithms, given the amount of information available; brute force if only one point needs to be interpolated (Section 3), octree search for groups of points (Section 4), and the fastest known vicinity algorithm (Section 5). These algorithms are combined in Section 6, yielding the fastest grid-to-grid algorithm, an advancing front vicinity algorithm. We then focus on the main innovations of the present paper: ways of improving robustness and speed by minimizing brute-force searches at corners and edges, vectorization of the interpolation procedure, and techniques to interpolate properly volumetric data separated by thin surfaces. Section 9 presents some timings, showing the considerable speedups obtained by the proposed approach. Finally, some conclusions are drawn.

2. THE BASIC ALGORITHM

Consider an unstructured finite element or finite volume mesh, as well as a point p with coordinates $\mathbf{x}_p$. A straightforward way to determine if the point p is inside a given element el is to determine the shape-function values of p with respect to the coordinates of the points belonging to el:

$$\mathbf{x}_p = \sum_i N^i \mathbf{x}_i. \tag{1}$$

0021-9991/95 $6.00

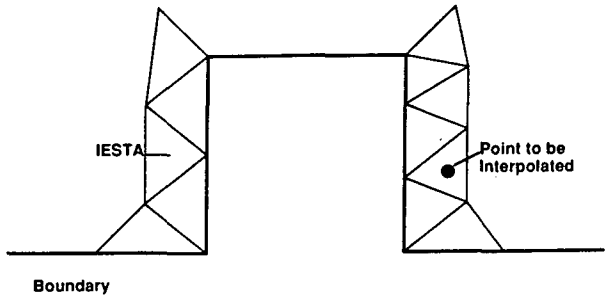


FIG. 5. Failure of nearest neighbour search algorithm.

outperform all other ones. The neighbour-to-neighbour search algorithm may be summarized as follows:

N.0. Form the List of Elements Adjacent to Elements for The Given Mesh;

N.1. DO: Loop over the points to be interpolated
N.2. Obtain good starting element START_ELEMENT;
N.3. For START_ELEMENT: Evaluate N^i from Eq. (4);
N.4. IF: Criterion (5) is satisfied THEN
Exit
ELSE
Set: START_ELEMENT to neighbour associated with $\min(N^i)$;
GOTO N.3
ENDIF
ENDDO

The neighbour-to-neighbour algorithm performs very well in the domain, but it can have problems on the boundary. Whereas the brute-force and octree search algorithms can "jump" over internal or external boundaries, the neighbour-to-neighbour algorithm can stop there (see Fig. 5). Its performance depends heavily on how good a guess the starting element START_ELEMENT is; it can be provided by bins, octrees, or alternate digital trees. On the other hand, due to its scalar nature, such an algorithm will not be able to compete with the octree search algorithm described in Section 3. Its main use is for point-to-grid or grid-to-grid transfer, where a very good guess for START_ELEMENT may be provided. This fastest grid-to-grid interpolation technique is described in the next section.

6. FASTEST GRID-TO-GRID ALGORITHM: VECTORIZED ADVANCING-FRONT VICINITY

The crucial new assumption made here, as opposed to all the other interpolation algorithms described so far, is that the points to be interpolated belong to a grid and that the grid connectivity (e.g., the points belonging to each element) is given as input. In this case, whenever the element END_ELEMENT of the known grid into which a point of the unknown grid falls is found, all the surrounding points of the unknown grid that have not yet been interpolated are given as a starting guess END_ELEMENT and stored in a list of "front" points LIST_FRONT_POINTS. The next point to be interpolated is then drawn from this list, and the procedure is repeated until all points have been interpolated. The procedure is sketched in Fig. 6, where the notion of "front" becomes apparent. The complete algorithm may be summarized as follows:

A.1. Form the list of elements adjacent to elements for the given mesh;
A.2. Form the list of points surrounding points for the unknown grid;
A.3. Mark points of the unknown grid as untouched
A.4. Initialize list of front points LIST_FRONT_POINTS for unknown grid
A.5. DO: For every non-interpolated point NON_INTERP_POINT
A.5. From LIST_FRONT_POINTS:
A.6. Obtain starting element START_ELEMENT in known grid
A.7. Attempt nearest neighbour search for NTRY attempts;
— IF unsuccessful: use brute force

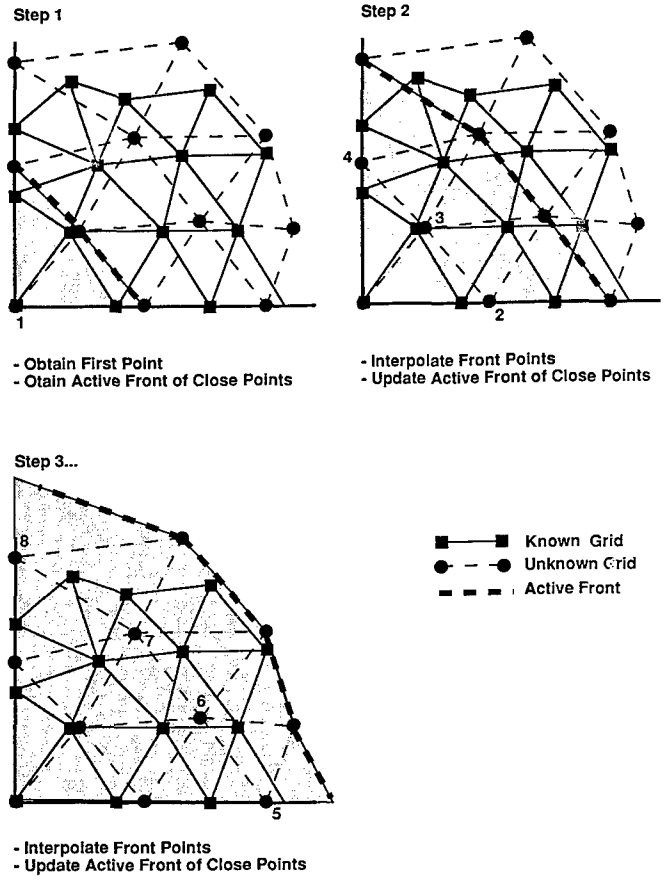


FIG. 6. Advancing front vicinity algorithm.

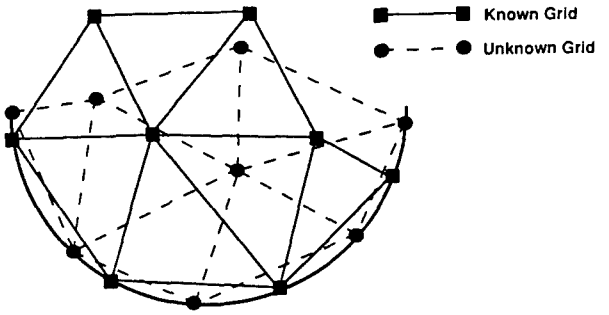


FIG. 7. Problems at concave boundaries.

— IF unsuccessful: stop or skip
⇒ END_ELEMENT

A.8. Store shape-functions and host elements

A.9. Loop over points surrounding NON_INTERP_POINT:
— IF: point has not been marked:
— Store END_ELEMENT as starting element for this point;
— Include this point in front LIST_FRONT_POINTS;
ENDIF

A.10. Mark point NON_INTERP_POINT as interpolated
ENDDO

A.11. IF: LIST_FRONT_POINTS not empty: GOTO A.5

Several possible improvements for this algorithm, layering of brute-force searches, inside-out interpolation, and vectorization, are detailed in the following.

6.1. *Layering of Brute-Force Searches*

In most instances (the exception being grids with very large disparity in element size where NTRY attempts are not sufficient), the neighbour-to-neighbour search will only fail on the boundary. Therefore, whenever a brute-force search is required, it is advisable to test first the elements connected to the boundary. This will reduce the brute-force search times considerably. Note, however, that we have to know the boundary points in this case. In the present case, the elements of the known grid are renumbered in such a way that all elements with three or more nodes on the boundary in 3D and two or more nodes on the boundary in 2D appear at the top of the list. These NR BOUNDARY ELS < NELEM elements are scanned first whenever a brute-force search is required. Moreover, after a front has been formed, only these elements close to boundaries are examined whenever a brute-force search is required.

6.2. *Inside-Out Interpolation*

This improvement is directed towards complex boundary cases. We group under this category cases where the boundary has sharp concave corners or ridges, or those cases where, due to the concavity of the surface points, the boundary may be close but outside of the known grid (see Fig. 7). In this case, it is advisable to form two front lists, one for the interior points and one for the boundary points. The interpolation of all the interior points is attempted first, and only then are the boundary points interpolated. This procedure reduces drastically the number of brute-force searches required for the complex boundary cases listed above. This may be seen from Fig. 8, where the brute-force at the corner was avoided by this procedure. As before, knowledge of the boundary points is required for this improvement.

6.3. *Vectorization*

The third possible improvement is vectorization. The idea is to search for all the points on the active front at the same time. It is not difficult to see that for large 3D grids, the vector-lengths obtained by operating in this manner are considerable. leading to very good overall performance. To obtain a vectorized algorithm we must perform steps N.3, A.7 as described above in vector mode executing the same operations on as many uninterpolated points as possible. The obstacle to this approach is that not every point will satisfy criterion (5) in the same number of attempts or passes over the points to be interpolated. The solution is to reorder the points to be interpolated after each pass such that all points that have as yet not

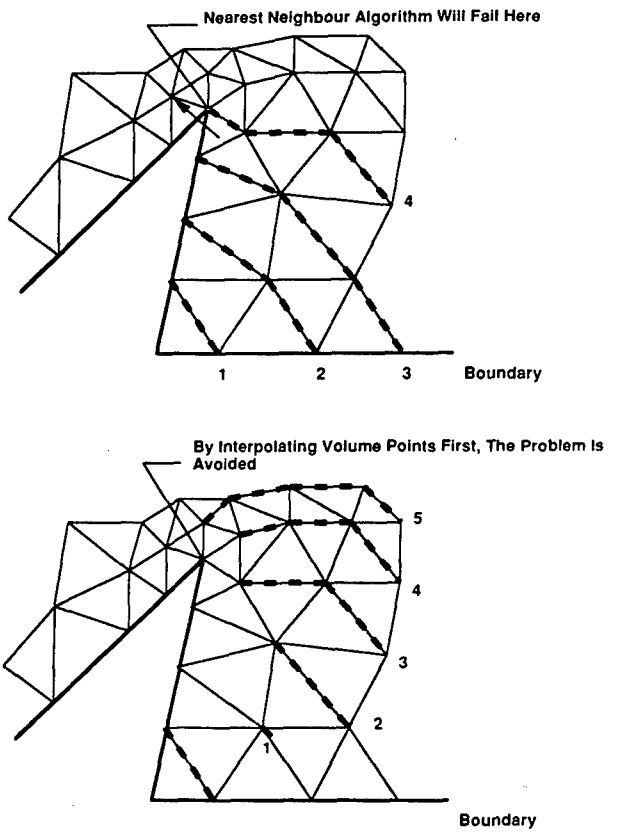


FIG. 8. Avoiding brute-force searches during interpolation.

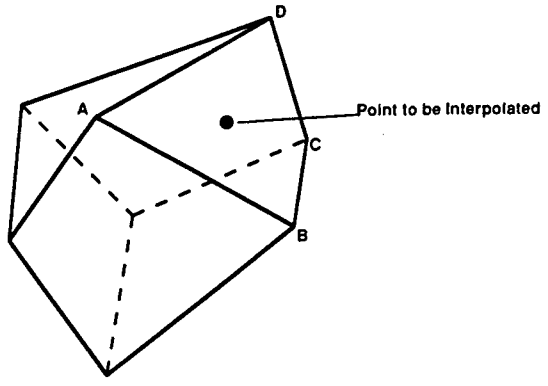


FIG. 1. Possible non-uniqueness for interpolation on bricks.

For triangles in 2D and tetrahedra in 3D, we have, respectively, two equations for three shape-functions and three equations for four shape-functions. The sum-property of shape-functions,

$$\sum_i N^i = 1, \tag{2}$$

yields the missing equation, making it possible to evaluate the shape-functions from the following system of equations:

$$\begin{Bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{Bmatrix} = \begin{Bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \\ 1 & 1 & 1 \end{Bmatrix} \cdot \begin{Bmatrix} N^1 \\ N^2 \\ N^3 \\ N^4 \end{Bmatrix}, \tag{3}$$

or, in concise matrix notation,

$$\mathbf{x}_p = \mathbf{XN} \rightarrow \mathbf{N} = \mathbf{X}^{-1}\mathbf{x}_p. \tag{4}$$

Then, the point p is in element el iff

$$\min(N^i, 1 - N^i) \geq 0, \quad \forall i. \tag{5}$$

For other types of elements more nodes than equations are encountered. The easiest way to determine if a point is inside an element is to split the element into triangles or tetrahedra and evaluate each of these sub-elements in turn. If the point happens to be in any of them, it is inside the element. This procedure may not be unique for highly deformed bricks, as shown in Fig. 1. Depending on how the diagonals are taken for the face A-B-C-D, the point to be interpolated may or may not be inside the element. Therefore, subsequent iterations may be required for bricks or higher-order elements with curved boundaries. Other ways to determine if a point is inside a bilinear element may be found in [6].

In the following, we will use the algorithm outlined above for triangles and tetrahedra as the starting point for improvements in performance. These improvements depend on the assumptions one can make with respect to the grids employed and the information available.

3. FASTEST 1-TIME ALGORITHM: BRUTE FORCE

Suppose we only have a given grid and a single point p with coordinates $\mathbf{x}_p$. The simplest way to find the element into which point p falls is to perform a loop over all the elements, evaluating their shape-functions with respect to $\mathbf{x}_p$:

```
—DO: Loop over all the elements
       — Evaluate N^i from Eq. (4);
       — IF: Criterion (5) is satisfied:
                Exit
         ENDIF
ENDDO
```

Because the central loop over all the elements can readily be vectorized this algorithm is extremely fast. We will use it in more refined algorithms both as a start-up procedure, as well as a fall-back position.

4. FASTEST N-TIME START ALGORITHM: OCTREE SEARCH

Suppose that, as before, we only have a given grid, but, instead of just one point p, a considerable number of points has to be interpolated. In this case, the brute-force algorithm described before will possibly require a complete loop over the elements for each point to be interpolated, and, on average, a loop over half the elements. A significant improvement in speed may be realized by only checking the elements that cover the immediate neighbourhood of the point to be interpolated. A number of ways can be devised to determine the neighbourhood (see Fig. 2):

—Bins, i.e., the superposition of a cartesian mesh [7, 8],

—Octrees, i.e., the superposition of an adaptively refined cartesian mesh [9, 10], and

—Alternate digital trees [11].

We consider octrees here, as bins perform poorly for problems where the nearest-neighbour distances vary by more than two orders of magnitude in the domain. One may form an octree with the element centroids or points. In the present case, we chose the latter option, as for tetrahedral grids the number of points is significantly less than the number of elements. The octree search algorithm then proceeds as follows:

— Form the octree for the points of the given mesh;
— Form the list of elements surrounding points for the given mesh;

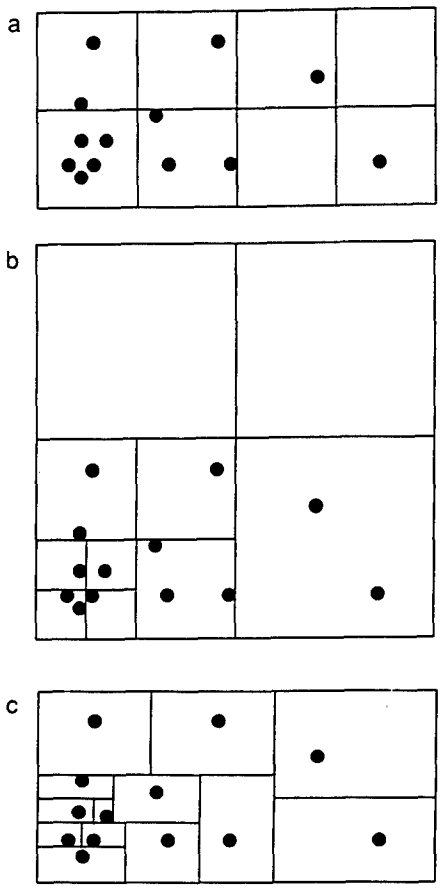


FIG. 2. Possible ways of subdividing space: (a) bins; (b) quadtree (octtree); (c) alternate digital tree.

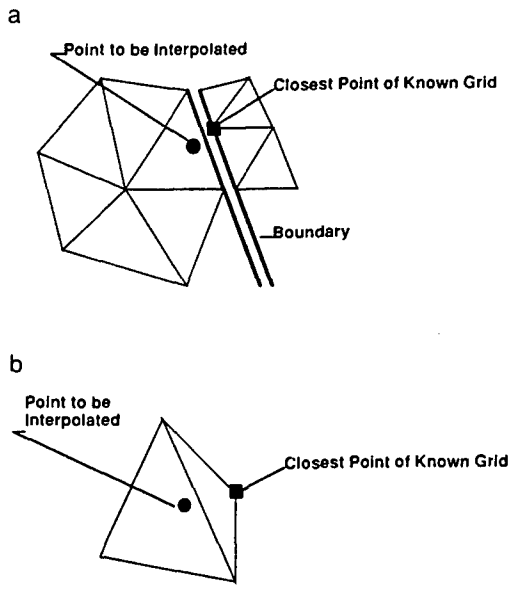


FIG. 3. Possible problems with closest point algorithm: (a) boundary gap: (b) distorted elements.

```
— DO: Loop over the points to be interpolated
      — Obtain close points of given mesh from the octree;
      — Obtain the elements surrounding the close points;
      — DO: Loop over the close elements:
              Evaluate N^i from Eq. (4);
              IF: Criterion (5) is satisfied:
                  Exit
              ENDIF
        ENDDO
— IF: We have failed to find the host element:
      Use brute-force over the elements
      ENDDO
```

Several improvements are possible for this algorithm. One may, in a first pass, evaluate the closest point of the given mesh to $\mathbf{x}_p$ and only consider the elements surrounding that point. Should this pass, which in general is successful, fail, the elements surrounding all the close points are considered in a second pass. Should this second pass also fail (see Fig. 3 for some pathological cases), one may either enlarge the search region, or use the brute-force algorithm described above in Section 2. The octree search algorithm is scalar for the first (integer) phase (obtaining the close points and elements), but all other stages may be vectorized. The vector lengths obtained for 3D grids are generally between 12 and 50, i.e., sufficiently long for good performance.

5. FASTEST KNOWN VICINITY ALGORITHM: NEIGHBOUR-TO-NEIGHBOUR

Suppose that, as before, we only have a given grid and a considerable number of points need to be interpolated. Moreover, assume that for any given point to be interpolated, an element of the known grid that is in the vicinity is known. In this case, it may be faster to jump from neighbour to neighbour in the known grid, evaluating the shape-function criterion [12] (see Fig. 4). If the element into which $\mathbf{x}$ falls can be found in a few attempts (<10), this procedure, although scalar, will

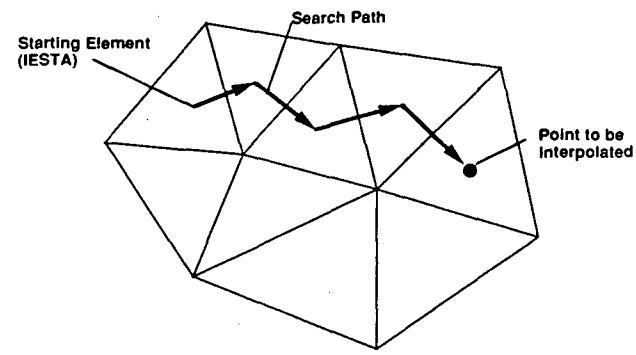


FIG. 4. Nearest neighbour jump algorithm.

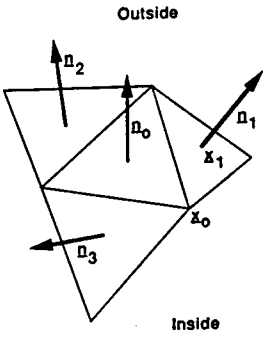


FIG. 9. Measuring surface concavity.

found their host element are at the top of the list. Such an algorithm proceeds in the following fashion:

V.0. Set the remaining number of points NR_REMAINING_POINTS=NR_FRONT_POINTS, where NR_FRONT_POINTS is the total number of points to be interpolated on the current front.

V.1. Perform steps N.3, A.7 in vector mode for all remaining points NR_REMAINING_POINTS.

V.2. Write the NR_NEXT_POINTS points that do not satisfy criterion (5) into a list LIST_OF_CURRENT_POINTS (1:NR_NEXT_POINTS). If NR_NEXT_POINTS=0: stop.

V.3. Write the NR_REMAINING_POINTS − NR_NEXT_POINTS points that do satisfy criterion (5) into LIST_OF_CURRENT_POINTS(NR_NEXT_POINTS +1 : NR_REMAINING_POINTS).

V.4. Reorder all point arrays using LIST_OF_CURRENT_POINTS. In this way, all points that have not yet found their host element are at the top of their respective lists (locations 1 : NR_NEXT_POINTS).

V.5. Set NR_REMAINING_POINTS=NR_NEXT_POINTS and go to V.1.

One can reduce the additional memory requirements associated with indirect addressing by breaking up all loops over the NR_REMAINING_POINTS remaining points into subgroups. This is accomplished automatically by using scalar temporaries on register to register machines. For memory to memory machines, a user-specified maximum group vector length must be specified.

7. CONCAVE SURFACES

For concave surfaces, criterion (12.5) will not be satisfied for a large number of surface points, prompting many brute-force searches. The algorithmic complexity of the interpolation procedure could potentially degrade to $O(N_b^2)$, where N_b is the number of boundary points. A considerable reduction of brute-force searches may be attained if the concavity of the surface can be measured. Assuming the unit face-normals **n** to be directed away from the domain, a possible measure of concavity is the visibility of neighbouring faces from any given face. With the notation of Fig. 9, the concavity of a region along the boundary may be determined by measuring the normal distance between the face and the centroids of the neighbouring faces. The allowable distance from the face for points to be interpolated is then given by some fraction α of the minimum distance measured:

$$d = \alpha|\min(0, \mathbf{n} \cdot (\mathbf{x}_0 - \mathbf{x}_i))|. \tag{6}$$

Typical values for α are $0.5 < \alpha < 1.5$. If a neighbour-to-neighbour search ends with a boundary face and all other shape-functions except the minimum satisfy Eq. (5), the distance of the point to be interpolated from the face is evaluated. If this distance is smaller than the one given by Eq. (6), the point is accepted and interpolated from the current element. Otherwise, a brute force search is conducted. The application of this procedure requires some additional arrays, such as face-arrays, a distance-array to store the concavity, and the relation between element faces and the face-array.

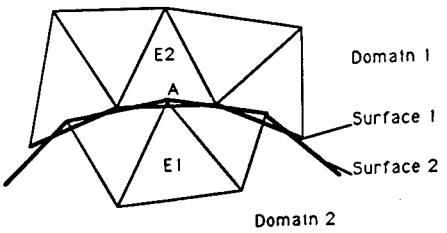


FIG. 10. Thin surface separating volumetric data.

8. VOLUMETRIC DATA SEPARATED BY THIN SURFACES

The interpolation of volumetric data for regions separated by thin surfaces is commonly encountered in computational physics. Examples for problems of this kind are flow simula-

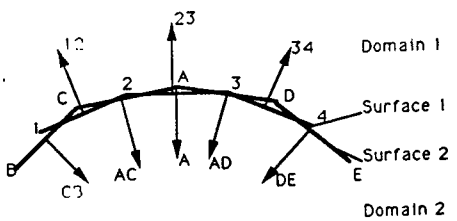


FIG. 11. Comparison of face and point normals. Note. IJ: normal of face IJ; I: normal of point I.

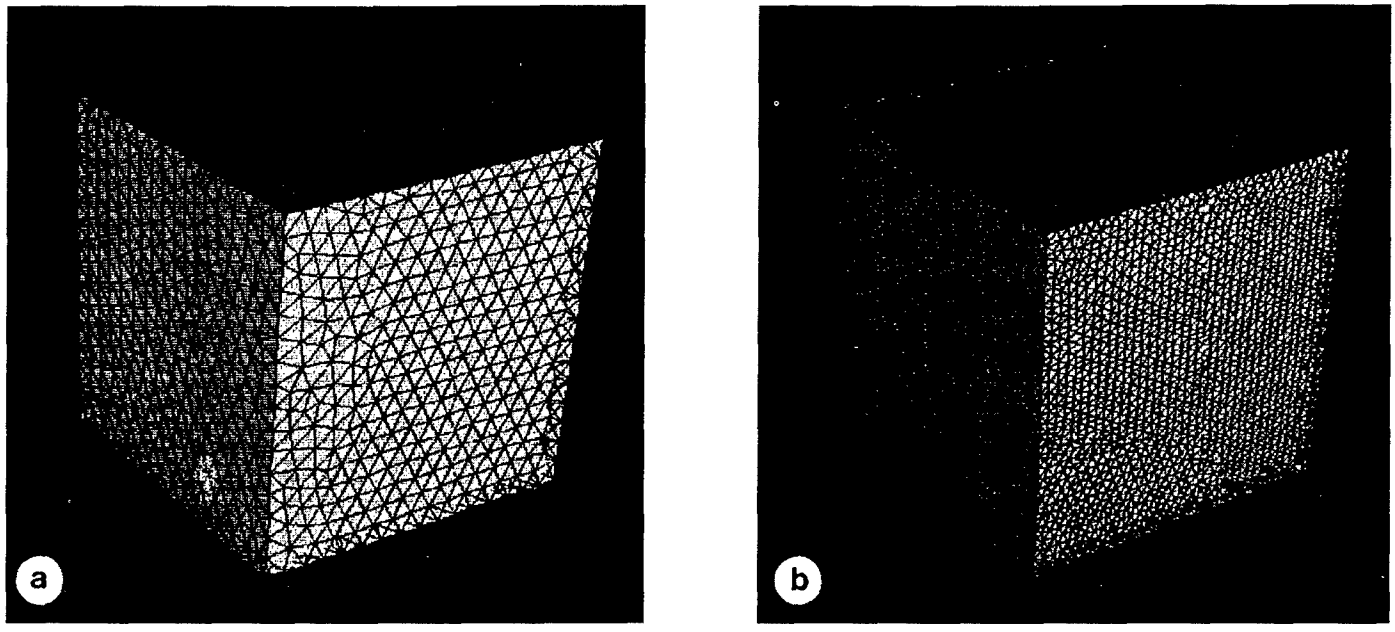


FIG. 12. Surface grids for a cube: NELEM = 34,661 (left); NELEM = 160,335 (right).

tions with thin separating sheets, such as trailing edges of wings, parasols, sails, airbags, shells, and others. In many of these cases, the surface points belonging to one of the two sides may lie inside an element that is attached to the other side. The situation is sketched in Fig. 10. Point A, although inside element E1, i.e., satisfying Criterion 5, should be interpolated from element E2. In order to avoid such an erroneous interpolation, the surface normals of the faces of the known grid are compared with the point normals of the points to be interpolated (see Fig. 11). If the scalar product of these normals falls below a preset tolerance (e.g., -0.5), the host element is rejected, and a brute search is performed. The surface point normals are obtained by averaging the normals of the faces surrounding them. While averaging, a comparison of the normals for all the surrounding faces is conducted. If these normals differ substantially, an edge or corner is detected, and the points are marked accordingly. For these points, the surface normal is considered as undefined, and no comparison of surface normals is conducted. The alignment test for surface normals just described can be

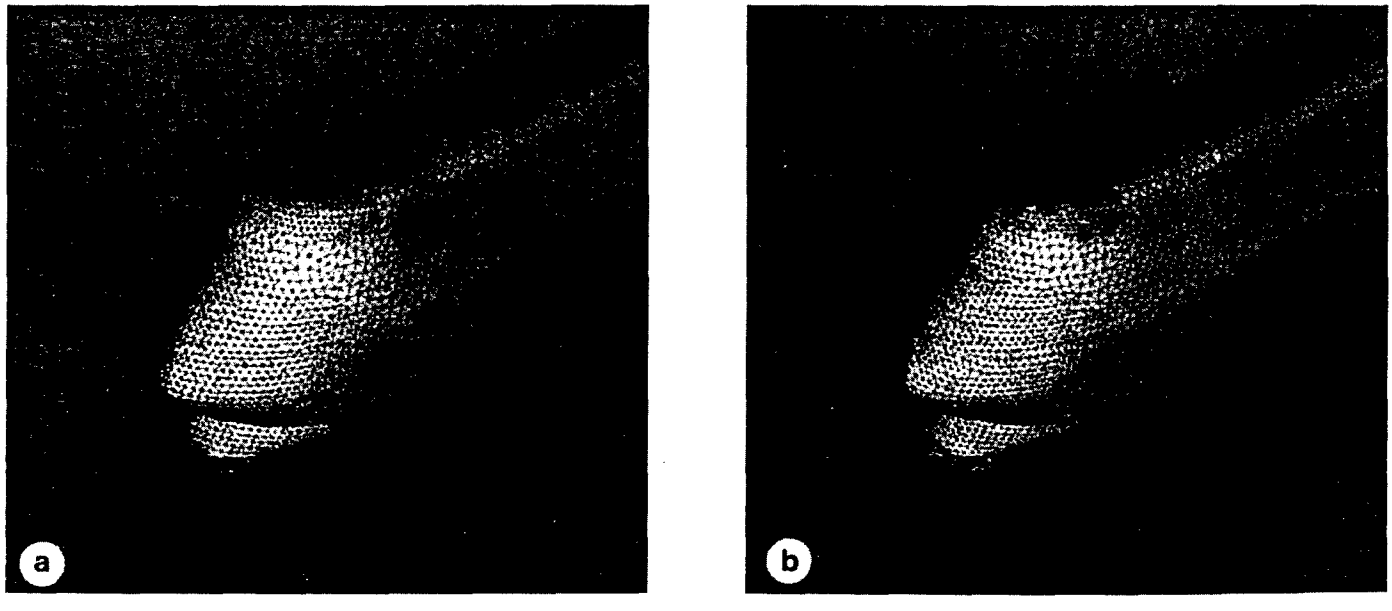


FIG. 13. Surface grids for a train: NELEM = 180,670 (left); NELEM = 243,068 (right).

APPENDIX 5: FLUID-STRUCTURE INTERACTION

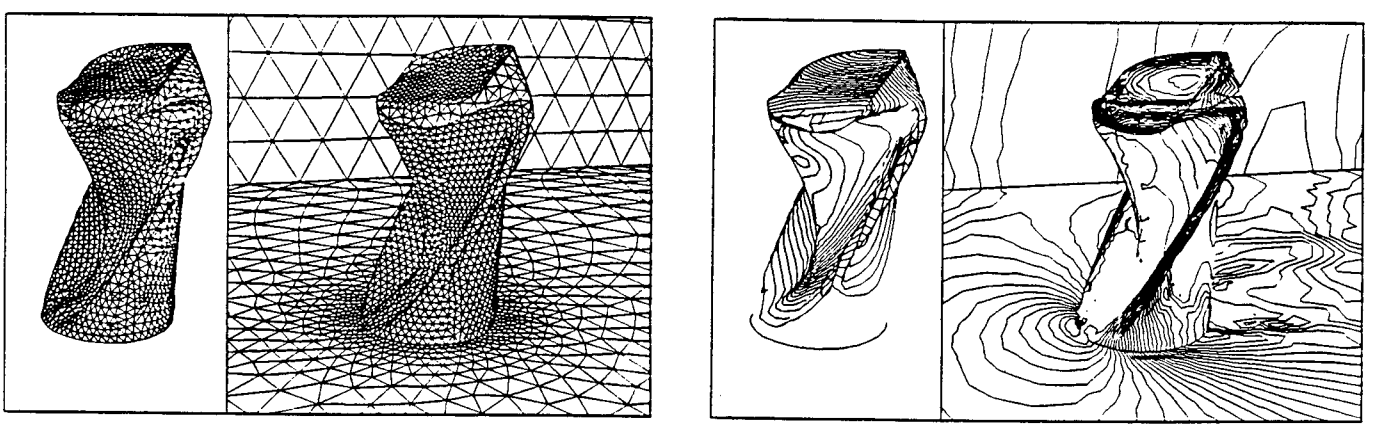
AIAA-95-2259

Fluid-Structure Interaction Using A Loose Coupling Algorithm And Adaptive Unstructured Grids

Rainald Löhner, Chi Yang and Juan Cebral
Institute for Computational Sciences and Informatics
George Mason University, Fairfax, VA 22030

Joseph D. Baum and Hong Luo
1710 Goodridge Drive, MS 2-3-1, McLean, VA 22102

Daniele Pelessone and Charles Charman
General Atomics, San Diego, CA 92121



SHOCK-CYLINDER INTERACTION (T=1.41E-03)

26th AIAA Fluid Dynamics Conference
June 19-22, 1995/San Diego, CA

FLUID-STRUCTURE INTERACTION USING A LOOSE COUPLING ALGORITHM AND ADAPTIVE UNSTRUCTURED GRIDS

Rainald Löhner[1], Chi Yang[1], Juan Cebral[1],
Joseph D. Baum[2], Hong Luo[2],
Daniele Pelessone[3] and Charles Charman[3]

[1]GMU/CSI, George Mason University, Fairfax, VA 22030, USA
[2]Science Applications International Corporation
1710 Goodridge Drive, MS 2-3-1, McLean, VA 22102, USA
[3]General Atomics, San Diego, CA 92121, USA

ABSTRACT

We present a loosely coupled algorithm to combine Computational Fluid Dynamics (CFD) and Computational Structural Dynamics (CSD) codes in order to solve, in a cost-effective manner, fluid-structure interaction problems. The basic fluid and structural dynamics codes are altered as little as possible. The structure is used as the 'master-surface' to define the extent of the fluid region, and the fluid is used as the 'master-surface' to define the loads. The transfer of loads, displacements, and velocities is carried out via fast interpolation and projection algorithms. As shown, this fluid-structure algorithm can be interpreted as an iterative solution to the fully-coupled, large matrix problem that results from the discretization of the complete problem. Results from practical shock-structure interaction problems indicate that the proposed approach offers a convenient and cost-effective way of coupling CFD and CSD codes without a complete re-write of them.

1. INTRODUCTION

Both Computational Fluid Dynamics (CFD) and Computational Structural Dynamics (CSD) have reached a high degree of reliability for the simulation of practical engineering problems. This has in turn led to widespread acceptance and an increase in user-friendliness for the codes most often used [1]. There exist large classes of important engineering problems that require the concurrent application of CFD and CSD techniques. Some examples are:

- Deformation or inflation of fabrics (parachutes, airbags, parasols, tents, etc.),
- Aeroelasticity of flexible structures (thin, high-aspect ratio wings, missiles, drones, etc.), where the deformation due to aerodynamic forces is such that significant changes in the flowfield are induced, leading to different loads.
- Shock/Structure Interaction, where the deformation of the structure may change the flowfield and the corresponding loads,
- Hypersonic Flight, where the deformation of the structure due to aerodynamic and aerothermal loads is such that a significant variation of the flowfield takes place (shock location, surface heating, etc.), and
- Variable Geometry Vehicles, where the change of geometry implies a transient phase in which structures and flowfields are interacting strongly, and, in most cases, non-linearly.

Most of these problems are presently solved either iteratively, i.e. making several cycles of 'CFD run followed by CSD run', or by assuming that the CFD and CSD problem can be decoupled 'to first order'. In most of the airframe manufacturing companies, as well as the shipyards, the respective CFD and CSD runs are performed in different divisions, leading to time-delays, loss of information, and, most importantly, loss of insight.

The need to solve fluid-structure interaction problems has prompted a number of developments in this field in recent years. The best way to sort these efforts is by classifying them according to the physical and numerical complexity employed for the fluid and structure respectively (see Figure 1). For the fluid, the PDEs solved are, in increasing order of physical complexity:

F1. Laplace/Helmholtz Operators (inviscid, irrotational , isentropic flow),
F2. Non-Linear Laplace Operators (inviscid, irrotational flow),
F3. Euler Equations (inviscid flow),
F4. Reynolds-Averaged Navier-Stokes Equations (viscous, time-averaged flow),
F5. Large-Eddy Simulations (viscous flow with spatio-temporal cut-off), and
F6. Navier-Stokes Equations.

Each of these approximations requires between one and two orders of magnitude more CPU-time and memory than the preceding one. For the linear case, boundary element methods may be employed, whereas all other approximations are typically approximated on a grid with spatial discretizations obtained from Finite Difference, Finite Volume, Finite Element, or Spectral Element techniques.

For the structure, the PDEs solved are, in increasing order of physical complexity:

S1. 6 Degrees of Freedom Integration (rigid body),
S2. Linear Elastic Models, either through
  a) A Modal Decomposition, or
  b) A Finite Element Discretization,
S3. Elasto-Plastic Models, and
S4. Elasto-Plastic Models with Contact, Rupture, etc.

As before, each of these approximations requires between one and two orders of magnitude more CPU-time and memory than the preceding one. For structures, the spatial discretization is typically carried out using Finite Element techniques [2].

A major characteristic of fluid-structure algorithms is the requirement to combine the discretizations for the fluid and the structure. This provides a third classification item (see Figure 2):

T1. Same surface discretization:
T2. Different surface discretization coupled via:
  a) Interpolation,
  b) Least-Squares,
  c) Lagrange Multipliers,
  d) A Third, so-called 'Virtual' Surface Grid.

For the simple CSD approximations S1,S2a, there is no discretization of the structure *per se*, so that the transfer of information between fluid and structure is straightforward.

With this series of possibilities, we are now in a position to classify previous fluid-structure interaction work. The two classic fields of structural acoustics and aeroelasticity have seen the largest amount of activity, particularly in those instances where the fluid and the structure were assumed as linear (inviscid, irrotational, isentropic fluid, and linear elastic structure). Of the many references, we mention:

F1-S2b-T1: see Everstine [3,4]
F1-S2b-T1: see Jackson and Christie [5]
F2-S2a: see Batina et al. [6]
F4-S1: see, e.g. Alonso et al. [7]
F3-S2a: see Guruswamy [8], Rausch et al. [9]
F3-S2b-T1: see Boschitsch and Quackenbush [10]
F4-S2b-T1: see Felker [11]
F4-S2b-T2d: see Guruswamy and Byun [12]

The present effort is directed towards practical nonlinear applications, in particular structures that undergo severe deformations due to aerodynamic or aero-thermodynamic loads. For this reason, we start immediately with the Euler and Reynolds-averaged Navier-Stokes equations for the fluid, and the nonlinear, large-deformation equations for the structure. Given that the geometrical complexity of the problems targeted for simulation can be severe, and the deformation considerable, automatic grid generation is a prime requirement. For this reason, unstructured grids are employed for both the fluid and the structure. The elements used for the fluid are tetrahedral, whereas the elements for the structure are typically bricks.

The remainder of the paper is organized as follows: Section 2 describes the coupling strategy used. The main layout of a code based on the loose coupling algorithm is described in Section 3. The individual codes chosen, **FEFLO96** for the fluid, and **DYNA3D** for the solid region, are briefly described in Section 4. Sections 5-8 discuss fast interpolation, unwrapping of doubly defined faces, surface tracking, and load transfer techniques. In Section 9, some demonstration runs are shown. Finally, conclusions and an outlook for future development are given in Section 10.

2. COUPLING ALGORITHM

When trying to compare the possible coupling algorithms, it is useful to start from the basic discrete equation systems obtained for the solid and fluid regions. For the solid region, we obtain, from a given Finite Element discretization, a system of equations of the form:

$$\mathbf{M}_s \frac{d\mathbf{v}_s}{dt} + \mathbf{D}\mathbf{v}_s + \mathbf{K}\mathbf{u} = \mathbf{f} \quad , \tag{1}$$

where $\mathbf{M}_s, \mathbf{v}_s, \mathbf{D}, \mathbf{K}, \mathbf{u}, \mathbf{f}$ denote, respectively, the mass-matrix, velocity vector, damping matrix, stiffness matrix, displacement vector and the loads vector. By splitting the degrees of freedom into those touching the fluid region ('sf'), and the remaining ones, we obtain

$$\begin{Bmatrix} \mathbf{M}_{sf} & 0 \\ 0 & \mathbf{M}_s \end{Bmatrix} \cdot \frac{d}{dt} \begin{pmatrix} \mathbf{v}_{sf} \\ \mathbf{v}_s \end{pmatrix} =$$

$$\begin{pmatrix} \mathbf{f}_{sf} \\ \mathbf{f}_s \end{pmatrix}^i + \begin{pmatrix} \mathbf{f}_{sf} \\ \mathbf{f}_s \end{pmatrix}^e + \begin{pmatrix} \mathbf{L} \cdot \mathbf{s}_{fs} \\ 0 \end{pmatrix} \quad , \tag{2}$$

where the superscripts i, e denote internal (stiffness, damping) and external forces respectively, $\mathbf{L}$ is the load matrix and $\mathbf{s}_{fs}$ the fluid stresses (pressures, shear stresses) on the surface. For the fluid region, we obtain, from a given Finite Element discretization, a system of equations of the form:

$$\begin{Bmatrix} \mathbf{M}_f & 0 & 0 & 0 \\ 0 & \mathbf{M}_f & 0 & 0 \\ 0 & 0 & \mathbf{M}_f & 0 \\ 0 & 0 & 0 & \mathbf{M}_{fs} \end{Bmatrix} \cdot \frac{d}{dt} \begin{pmatrix} \rho \\ \mathbf{s} \\ \mathbf{v}_f \\ \mathbf{v}_{fs} \end{pmatrix} =$$

$$\begin{pmatrix} \mathbf{f}_\rho \\ \mathbf{f}_p \\ \mathbf{f}_v \\ \mathbf{f}_{v_{fs}} \end{pmatrix}^i + \begin{pmatrix} \mathbf{f}_\rho \\ \mathbf{f}_p \\ \mathbf{f}_v \\ \mathbf{f}_{v_{fs}} \end{pmatrix}^e , \qquad (3)$$

where, for the sake of clarity, we have employed the non-conserved variables: density, velocities and pressure $(\rho, \mathbf{v}, p)$, and the discrete degrees of freedom have been separated into those that touch the solid ('fs') and the rest. $\mathbf{M}_f, \mathbf{f}_\rho, \mathbf{f}_p, \mathbf{f}_v$ denote, respectively, the mass-matrix, right-hand side vectors for the density, pressure, and velocities. The combined fluid-structure system now assumes the form

$$\begin{Bmatrix} \mathbf{M}_f & 0 & 0 & & 0 & 0 \\ 0 & \mathbf{M}_f & 0 & & 0 & 0 \\ 0 & 0 & \mathbf{M}_f & & 0 & 0 \\ 0 & 0 & 0 & \mathbf{M}_{fs} + \mathbf{M}_{sf} & & 0 \\ 0 & 0 & 0 & & 0 & \mathbf{M}_s \end{Bmatrix} \cdot \frac{d}{dt} \begin{pmatrix} \rho \\ \mathbf{s} \\ \mathbf{v}_f \\ \mathbf{v}_{fs} \\ \mathbf{v}_s \end{pmatrix} =$$

$$\begin{pmatrix} \mathbf{f}_\rho \\ \mathbf{f}_p \\ \mathbf{f}_{vf} \\ \mathbf{f}_{v_{fs}} + \mathbf{f}_{sf} \\ \mathbf{f}_s \end{pmatrix}^i + \begin{pmatrix} \mathbf{f}_\rho \\ \mathbf{f}_p \\ \mathbf{f}_{vf} \\ \mathbf{f}_{v_{fs}} + \mathbf{f}_{sf} \\ \mathbf{f}_s \end{pmatrix}^e + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathbf{L} \cdot \mathbf{s}_{fs} \\ 0 \end{pmatrix} , \qquad (4)$$

where we have set $\mathbf{v}_{sf} = \mathbf{v}_{fs}$ as required for Navier-Stokes applications. For Euler problems, we only require an equality of the normal velocities $\mathbf{v}_{sf}^n = \mathbf{v}_{fs}^n$. Given this complete system, we can now define possible coupling algorithms.

a) Tight coupling: We denote by tight coupling the simultaneous update of all variables, including (and most notably) those at the fluid/structure interface. This implies solving the complete system given by Eqn.(4) in one step. The formulation allows for different grids in the CFD and CSD domains, but the reader should realize that the derivation of the proper projection integrals can be tedious in 3-D. From a practical point of view, choosing this approach requires an almost complete re-write of the CFD and CSD codes into one single coupled code. This implies a loss of modularity, as well as the inability to couple one CFD code with several CSD codes (or vice-versa). Moreover, the 'trade-oriented' aspect of each of the individual codes is blurred or lost, with the associated extra expenses for retraining the user base.

b) Loose coupling: We denote by loose coupling the separate update of the CFD and CSD domains, with a transfer of variables at the interface. The most common way of realizing this approach is by selecting a 'master surface' for a certain variable, and interpolating or projecting the variable to the other domain at the beginning of the next timestep. For CFD/CSD problems, the most natural combination is to select the CSD surface location and velocity as the 'master-grid' for displacements, and the CFD grid as the 'master-grid' for the loads (pressures, shear-stresses). The product of displacement times load yields work, making the combination physically appealing. This approach may be regarded as an iterative solution of the combined system given by Eqn.(4). Each iterative pass is composed of the following steps:

- Solve for CFD with imposed $\mathbf{v}_{sf}, d\mathbf{v}_{sf}/dt$;
- Solve for CSD with imposed $\mathbf{s}_{sf}$ and $\mathbf{M}_{fs} \cdot d\mathbf{v}_{sf}/dt$.

Note that unlike discretizations obtained from boundary integral methods, the error incurred by neglecting the added mass $\mathbf{M}_{fs} + \mathbf{M}_{sf}$ is very small, as these terms only contain contributions from the elements adjacent to them. For an air/steel interface, the ratio of densities is $O(10^3)$, for water/steel $O(10)$.

Depending on the time integration scheme used for the CSD and CFD domains, several simplifying strategies can be employed. Should explicit time integration be the proper way to advance the CSD and CFD regions (as is the case for the class of problems considered here), the loose and tight coupling systems are almost identical, the only error being the mass of fluid for the first row of elements adjacent to the solid. Should implicit time integration be the proper way to advance the CSD and CFD regions (as is the case for low-frequency aeroelastic applications), the LHS of the time-discrete form of Eqn.(4) will contain entries of the Jacobians of $\mathbf{f}^i$. In this case, the iterative strategy discussed above will have to be used for the loose coupling approach if equivalency with the tight coupling system is to be achieved. Finally, if only a steady-state solution for the coupled fluid-structure system is sought, the loose coupling approach may be used either with explicit or implicit time integration for the CSD and CFD domains without incurring any errors.

The variables on the boundaries are transferred back and forth between the different codes by a master code that directs the multi-disciplinary run. Each code (CFD, CSD, CEM, ..) is seen as a subroutine, or object, that is called by the master code. This implies that the transfer of geometrical and physical information is performed between the different codes without affecting their layout, basic functionality, and coding styles. This is seen as the main advantage of this approach.

A tremendous amount of man-years has been devoted

to CFD and CSD codes, incorporating into them all the minor features that make these codes efficient, practical, user-friendly tools. The central assumption made here is that these codes will not be rewritten again, should be left alone in their present and future development, and nevertheless can be combined efficiently to solve strongly coupled CFD/CSD problems. For structures that break, rupture, or deform markedly due to the loads exerted by the fluid, the corresponding CFD and/or CSD grid will require some form of remeshing. This remeshing can either be local or global in nature. If this remeshing can not be done automatically, the usefulness of such an approach will always remain limited. Therefore, automatic gridding techniques are an enabling technology for this class of problems. The CFD code employed here has, as one of its salient features, an automatic remeshing capability. This capability is very important for the class of fluid-structure interaction problems considered, and will be demonstrated in the examples shown below.

3. APPROACH CHOSEN

As stated before, the loose coupling approach is followed here to bring together, in a general, cost effective way, CFD and CSD codes currently in widespread use, in order to solve strongly coupled CFD/CSD problems. The global timestepping algorithm, sketched in Figure 3, proceeds as follows:

- Set: `istar=1,istop=0`
- `call CFD-code(.., istar, ..)`
  - Read in CFD-Data
  - Initialize all CFD Arrays
- `call CSD-code(.., istar, ..)`
  - Read in CSD-Data
  - Initialize all CSD Arrays
- `if(istop.EQ.0) then`
  - `call CFD-code(.., istop, tends, tendf, ..)`
  - `call CSD-code(.., istop, tends, tendf, ..)`
- `endif`

Here `tendf, tends` denote the ending times for the CFD and CSD code respectively. The algorithm outlined above clearly leaves the possibility open to perform N-CFD-code steps, followed by M-CSD-code steps, i.e. asynchroneous timestepping. It is felt that this is of considerable importance in order to keep the algorithm as general as possible. As one can see, both the CFD and the CSD codes are called as subroutines. The argument list passed contains all the variables required for the inter-grid transfer of information. Having outlined the coupled procedure, we can now examine the individual domains more closely.

a) For the **CFD code**:

Variables passed in:
- Ending Time of CSD Step: `tends`
- Position/Velocity of Surface Points: `coobs, vpobs`

Then:
- Impose Surface Positions/Velocities from: `coobs, vpobs` at time `tendf`
- `if(time.lt.tends) then:`
  - Advance CFD Solution One Timestep
  - Update CFD Mesh
  - Refine/Remesh CFD Domain As Required
  - Update `time`
- `endif`
- Set `tendf=time`
- Compute Loads (Pressure, ..) for the CSD Surface Points
- `return`

Variables passed out:
- Ending Time of CFD Step: `tendf`
- Loads for the CSD Surface Points: `loabs`

b) For the **CSD code**:

Variables passed In:
- Ending Time of CFD Step: `tendf`
- Loads at the Wetted Points: `loabs`

Then:
- Impose Surface Loads from CFD Code
- `if(time.lt.tendf) then:`
  - Advance CSD Solution One Timestep
  - Update CSD Mesh Accordingly
  - Refine/Remesh CSD Domain As Required
  - Update `time`
- `endif`
- Set `tends=time`
- Compute Positions and Velocities for the Wetted (CFD) Surface Points
- `return`

Variables passed out:
- Ending Time of CSD Step: `tends`
- Position/Velocity of Surface Points: `coobs, vpobs`

4. CODES SELECTED

The selection of the respective CFD/CSD codes was made according to the following guidelines:
- The code must be well proven;
- The code must be benchmarked;
- The code must be supported;
- The code must have a user base/community;

The two candidate codes chosen were: **FEFLO96** for the fluid and **DYNA3D** for the solid. A brief

overview of the physics being modelled, the numerical techniques employed, as well as useful engineering, meshing and software options available in these two codes is given in the sequel.

4.1 CFD CODE: **FEFLO96**

a) Physics: **FEFLO96** is a simulation code for compressible flows. The equations solved are the Euler, Laminar or Reynolds-averaged Navier-Stokes equations, as well as the linear acoustics equations. The turbulence models available are the Smagorinsky and Baldwin-Lomax models, as well as a user-input option via subroutine. Work is in progress on the k-epsilon and k-omega models. Equations of state supported by **FEFLO96** include ideal polytropic gas, real air EOS table look-up, water EOS table look-up, and a link to the general SESAME library of EOS. In order to handle situations with moving bodies and/or moving grids, the equations are solved in the Arbitrary Lagrangean-Eulerian frame [13].
Flows with particles are treated via a second solid phase. The particles interact with the fluid, exchanging mass, momentum and energy, and are integrated in a time-consistent manner with the fluid.

b) Numerics: The spatial discretization is accomplished via finite element techniques on unstructured tetrahedral grids. In order to achieve high execution speeds, edge-based data structures are used. Both central and upwind flux (van Leer [14], Roe [15]) formulations are possible. For the temporal discretization, both Taylor-Galerkin and Runge-Kutta time integration schemes are available. Monotonicity of the solution may be achieved through a blend of second and fourth order dissipation [16], pressure-based, Flux-Corrected Transport (FCT) [17], or classic TVD limitors. The particles are integrated using a second-order Runge-Kutta scheme, and optimal tracking techniques [18] have been implemented to expedite the transfer of information between fields and particles.

c) Engineering: In order to handle situations with moving bodies, **FEFLO96** offers a variety of options: prescribed motion, 6-DOF integration based on aerodynamic forces, and link to CSD codes.
A variety of boundary conditions can be prescribed to simulate as faithfully as possible engineering flows: sub-,tran-, and supersonic in/outflow, total pressure inflow b.c., static pressure, mach-number and normal flux outflow b.c., porous walls, and periodicity. At the same time, a large variety of diagnostics is produced by the code to track or display specific parts of the flowfield that are of special interest: 0-D probes (e.g. for station time history), 1-D line segments for x/y display, 2-D planes or iso-surfaces for contouring, flux trough surfaces, force and moment data on surfaces or bodies, on-line display of the flowfield, etc.

d) Meshing Options: **FEFLO96** allows for automatic adaptive h-refinement [19] and automatic remeshing [20,21] in order to enhance the solution accuracy, even for situations with moving bodies.

e) Software: **FEFLO96** is written in FORTRAN-77 and fully vectorized. Renumbering techniques [22] are used extensively in order to avoid cache-misses on RISC-based machines. Parallelization is achieved via domain splitting [23]. The code runs on all major workstations, vector-supercomputers and parallel platforms.

FEFLO96 is a well-proven and benchmarked code used extensively by the authors and others in the CFD community [24-28].

4.2 CSD CODE: **DYNA3D**

a) Physics: **DYNA3D** is a simulation code especially suited for solids undergoing rapid and severe deformation. The conservation equations for momentum are written and solved for in the Lagrangian frame of reference. The large deformation, large strain formulation is employed throughout. The code incorporates forty-one different material models, among them linear elastic, linear elastic-plastic, strain-rate sensitive steel with fracture, hardening material models, a geological cap model for soil materials, and a variety of concrete models that simulate fracturing behavior [29]. **DYNA3D** also offers eleven equations of state models, including equations of state for high explosives.

b) Numerics: The spatial discretization is accomplished via finite element techniques on unstructured grids. The elements available for structural modelling are one truss and two beam elements, several quadrilateral shell elements (e.g. Belitschko-Tsai [30], Hughes-Liu [31], YASE [29], and QPII [32], and hexaheral elements with one-point integration for the 3-D solids. The shells allow for multiple integration points across the thickness, making it possible to accurately treat nonlinear plastic behavior of simple and composite shells. Several hourglass control options may be used. We have found that the Flanagan-Belitschko hourglass control [33] works best for the unstructured hexahedral grids we most often employ. The temporal discretization is carried out using an explicit central difference method, which is conditionally stable.

c) Engineering: **DYNA3D** incorporates a large number of convenient features that prove especially useful for realistic engineering problems. The following is a non-exhaustive list of those features that were particularly relevant to our class of applications. The user may prescribe non-reflecting boundary conditions which eliminate stress wave reflections at model

boundaries, making it possible to use smaller models. There are twelve types of sliding-interface algorithms to treat different interface conditions between interacting parts. Sliding-interface algorithms permit the treatment of contact conditions with friction, gap opening, spotwelds, etc. For civil engineering applications, there are rebar-concrete interaction algorithms which include degradation and failure of bond.
A large variety of diagnostics is produced by the code to track or display specific parts of the structure that are of special interest: 0-D probes (e.g. for station time history), 1-D line segments for x/y display, 2-D planes or iso-surfaces for contouring, stress, strain, force and moment data on surfaces or fields, etc.

d) Meshing Options: **DYNA3D** in its current state does not allow for automatic adaptive h-refinement or automatic remeshing. Work is currently in progress to incorporate these feature into **DYNA3D** .

e) Software: **DYNA3D** is written in FORTRAN-77 and fully vectorized. The code was written with CRAY-type machines in mind, but runs well on all major workstations, vector-supercomputers and some parallel platforms. It employs dynamic memory allocation, making it capable of solving very large problems.

DYNA3D is a well-proven and benchmarked code used extensively by the authors and others in the CSD community [29,34,35]. **DYNA3D** was developed at the Lawrence Livermore National Laboratories by Dr. John Hallquist with contributions from Dr. David Benson and Dr. Robert Whirley. **DYNA3D** has been successfully used for a large number of applications, including nuclear and conventional weapon design, car and airplane crashworthiness studies, analysis of reinforced structures such as bunkers, tunnels, and silos, as well as spent nuclear shipping cases. It is supported and maintained by Lawrence Livermore National Laboratories.

5. SURFACE TO SURFACE INTERPOLATION

One of the main aims of the proposed approach is to couple the different codes in such a way that each one of the codes used is modified in the least possible way. Moreover, the option of having different grids for different disciplines (CFD/CSD/CEM..), as well as adaptive grids that vary in time, implies that in most cases no fixed common variables will exist at the boundaries. Therefore, fast and accurate interpolation techniques are required. As the grids may be refined/coarsened during timesteps, and the surface deformations may be severe, the interpolation procedures have to combine speed with generality. In what follows, we will use the word **interpolation** as refering to the process of finding, from a list of faces, the one closest to a given point, and the word **evaluation** to the actual product of shape-functions with nodal values, e.g. in order to obtain the pressure at a point on a face.

Consider the problem of fast interpolation between two surface triangulations. Other types of surface elements can be handled by splitting them into triangles, so that what follows may be applied to such gridtypes as well. In the sequel, we will denote the triangular surface elements as faces. The basic idea is to treat the topology as 2-D, while the interpolation problem is given in 3-D space. This implies that further criteria, like relative distances normal to the surface, will have to be employed in oder to make the problem unique. The basic procedure is to compute the shape-functions of the surface triangles for any point being interpolated. Using the notation of Figure 4, we can write:

$$\mathbf{x}_p = \mathbf{x}_0 + \sum_{i=1,3} \alpha^i \mathbf{g}_i \quad , \tag{5}$$

where

$$\mathbf{g}_{1,2} = \mathbf{x}_{1,2} - \mathbf{x}_0 \quad , \quad \mathbf{g}_3 = \frac{\mathbf{g}_1 \times \mathbf{g}_2}{|\mathbf{g}_1 \times \mathbf{g}_2|} \quad , \tag{6a,b}$$

$$\alpha^{1,2} = N^{1,2} \quad , \quad \alpha^0 = 1 - \alpha^1 - \alpha^2 \quad . \tag{6c,d}$$

Point $\mathbf{x}_p$ may be considered as being on the surface face Iff:

$$min(N^i, 1 - N^i) \geq 0 \quad , \quad \forall i = 0, 1, 2 \quad , \tag{7a}$$

and

$$d_n = |\alpha^3 \mathbf{g}_3| \leq \delta_n \quad . \tag{7b}$$

Here δ_n denotes a tolerance for the relative distance normal to the surface face. Many search and interpolation algorithms have been devised over the years. We have found that for generality, a layered approach of different interpolation techniques works best. Wherever possible, a vectorized advancing front neighbour-to-neighbour algorithm is employed as the basic procedure [36]. Should this fail, octrees [37,38] are employed. Finally, if this approach fails too, a brute force search over all the surface faces is performed [36]. For realistic 3-D surface geometries, the interpolation of surface grid information may be complicated by a number of the factors. The first of these factors is the proper choice of δ_n, i.e. the proper answer to the question: 'How close must a face be to a point in order to be acceptable ?' This is not a trivial question for situations where narrow gaps exist in the CFD mesh, and when there is a large discrepancy of face-sizes between surface grids. Our experience indicates that for attached surface tracking, the choice

$$\delta_n < c_n \cdot |\mathbf{g}_1 \times \mathbf{g}_2|^{0.5} \quad , \quad c_n = 0.05 \quad , \qquad (8)$$

works reliably, although the constant c_n may be problem dependent. A second complication often encountered arises due to the fact that Eqn.(7a) may never be satisfied (e.g. the convex ridge shown in Figure 5a), or may be satisfied by more than one surface face (e.g. the concave ridge shown in Figure 5b). In the first instance the criterion given by Eqn.(7a) may be relaxed somewhat to

$$min(N^i, 1 - N^i) \geq \epsilon \quad , \quad \forall i = 0, 1, 2 \quad , \qquad (9)$$

where ϵ is a small number. For the second case, the surface face with the smallest normal distance d_n is selected. We remark that in both of these instances the interpolation error is unaffected by the final host surface face, as the interpolation weights are such that only the points belonging to the ridge are used for interpolation. However, not choosing the correct face may lead to CFD elements with negative volumes as CFD surface points are dragged to the incorrect CSD surface. We have found that it is very important to take the face that has the smallest distance to the point being interpolated in order to mitigate these problems. For situations close to corners, gaps, or multi-body configurations, an exhaustive search over all faces will be triggered. In order not to check in depth the complete surface mesh, only the faces that satisfy the relaxed closeness criteria $\epsilon \geq -1$, $c_n \leq 0.5$ are considered. The face with the closest distance to the point is kept. If a face satisfies Eqn.(7a), the closest distance is indeed d_n. Should this not be the case, the closest distance to the three edges ij of the face is taken:

$$\delta = min_{ij} \, |\mathbf{x}_p - (1 - \beta_{ij})\mathbf{x}_i - \beta_{ij}\mathbf{x}_j| \qquad (10.a)$$

$$\beta_{ij} = \frac{(\mathbf{x}_p - \mathbf{x}_i) \cdot (\mathbf{x}_j - \mathbf{x}_i)}{(\mathbf{x}_j - \mathbf{x}_i) \cdot (\mathbf{x}_j - \mathbf{x}_i)} \qquad (10.b)$$

Should two faces have the same normal distance, the one with the largest minimum shape-function α^i, $i = 0, 1, 2$ is retained.

A third complication arises for cases where thin shells are embedded in a 3-D volumetric fluid mesh. For these cases, the 'best' face may actually lie on the opposite side of the face being interpolated. This ambiguity is avoided by defining a surface normal, and then only considering the faces and points whose normals are aligned, i.e. those for which

$$\mathbf{n}_f \cdot \mathbf{n}_p > c_s \quad , \quad c_s = 0.5 \quad . \qquad (11)$$

Here $\mathbf{n}_f, \mathbf{n}_p$ denote the face and point-normals respectively. The definition of a proper point-normal from the face-normals can be problematic for non-smooth surface, such as those obtained when severe buckling or wrinkling occurs. For these cases, all faces are considered for interpolation.

Experience indicates that it is advisable to perform a local exhaustive search for all faces surrounding the best face found in order to obtain the face that satisfies Eqns.(7,10) as best possible.

6. UNWRAPPING DOUBLY DEFINED FACES

Consider the common case of thin structural elements, e.g. roofs, walls, stiffeners, etc. surrounded by a fluid medium. The structural elements will be discretized using shell elements. These shell elements will be affected by loads from both sides. Most CSD codes require a list of faces on which loads are exerted. This implies that the shell elements loaded from both sides will appear twice in this list. In order to be able to incorporate thickness and interpolate between CSD and CFD surface grids in a unique way, these doubly defined faces are identified, and new points are introduced. The first step is to identify the doubly defined faces. A linked list that stores the faces surrounding each point is constructed. Each face is then checked by performing an exhaustive comparison of the points of each of the faces surrounding the first node of each face. This will identify doubly defined faces in $O(N_f)$ complexity ,where N_f is the number of faces. Should this check reveal the existence of doubly defined faces, new points are introduced using an unwrapping procedure. A faces-surrounding-faces list `fsufa(nedfa,nface)` is built, where `nedfa` denotes the number of edges per face, and `nface` the number of faces. As any given face may have multiple neighbour faces across an edge, it is important to select the most suitable neighbour (see Figure 6). This is done by comparing the scalar products of the face-normals between neighbours, as well as the visibility of neighbour points from the current face. The rest of the procedure is best explained in the following pseudo-code form:

Initialize point array `lpoin(1:npoin)=0`

```
do iface=1,nface ! Loop over the faces
 do inofa=1,nnofa ! Loop over the face-nodes
 ipoin=bface(inofa,iface) ! Point number
 if(ipoin.gt.0) then
  if(lpoin(ipoin).eq.0) then
    The point has not yet been surrounded ⇒
    ipold=ipoin
    ipnew=ipoin
  else
    As the point has already been surrounded and
    the point was left unconsidered:
    introduce a new point
    ipold=ipoin
    npoin=npoin+1
    ipnew=npoin
    Transcribe coodinates and points of ipold
    to ipnew
  endif
  Surround the point with faces obtained from fsufa
  that have point ipold in common
  Modify bface, setting entry of ipold to -ipnew
  Mark the point as surrounded: lpoin(ipoin)=-1
 endif
 enddo
enddo
```

Restore `bface` to positive values.

The unwrapping of multiply defined faces is shown in Figure 7. Having unwrapped the surface mesh, a unique set of point-normals that point into the CFD field is obtained. Taking into consideration the thickness of each shell, the point coordinates used for interpolation onto the CFD surface are computed.

7. SURFACE TRACKING TECHNIQUES

An important question that needs to be addressed is how to make the different grids follow one another when deforming surfaces are present. Consider the typical aeroelastic case of a wing deforming under aerodynamic loads. For accuracy purposes, the CFD discretization will be fine on the surface, and the surface will be modelled as accurately as possible from the CAD-CAM data at the start of the simulation. On the other hand, a CSD discretization that models the wing as a series of plates may be entirely appropriate. If one would force the CFD surface to follow the CSD surface, the result would be a wing with no thickness, clearly inappropriate for an acceptable CFD result. On the other hand, for strong shock/object interactions with large plastic deformations and possible tearing, forcing the CFD surface to follow exactly the CSD surface is the correct way to proceed. These two examples indicate that more than one strategy may have to be used to interpolate and move the surface of the CFD mesh as the structure moves. We have incorporated the following techniques:

a) Exact tracking with linear interpolation. This is the most straightforward case, but, as could be seen from the example described above, may lead to bad results.

b) Exact tracking with quadratic interpolation. In this case, the surface normals are recovered at the end-points of the surface triangulation. For each edge of the triangulation, the midpoint is extrapolated using a Hermitian polynomial (see Figure 8). In this way, quadratic triangles are obtained. The surface is then approximated/interpolated using this higher order surface.

c) Tracking with initial distance vector. In many instances, e.g. thick shells, the CFD and CSD domains will never coincide. A way to circumvent this dilemma is to compute the difference vector between the initial CSD and CFD surfaces, and maintain this vector (allowing for translation and rotation) for the duration of the coupled run. Several options are possible here, and we are still actively exploring which is the best way to proceed.

An important area currently under investigation is how to handle, in an efficient and automatic way, models that exhibit incompatible dimensionalities. An example for such a 'reduced model' would be an aeroelastic problem where the wing structure is modeled by a torsional beam (perfectly acceptable for the lowest eigenmodes), and the fluid by a 3-D volumetric mesh. It is easy to see that the proper specification of movement for the CFD surface based on the 1-D beam, as well as the load transfer from the fluid to the beam, represent non-trivial problems for a general, user-friendly computing environment.

8. CFD-CSD LOAD TRANSFER

During each global cycle, the CFD loads have to be transferred to the CSD mesh. Simple point-wise interpolation can be employed for those cases in which the elements of the CSD surface mesh are smaller or of similar size than the elements of the CFD surface mesh. However, this approach is not conservative, and will not yield accurate results for the common case of CSD surface elements being larger than their CFD counterpart. Considering without loss of generality the pressure loads only, it is desirable to attain:

$$p_s(\mathbf{x}) \approx p_f(\mathbf{x}) \quad , \tag{12}$$

while being conservative in the sense of:

$$\mathbf{f} = \int p_s \mathbf{n} d\Gamma = \int p_f \mathbf{n} d\Gamma \quad , \tag{13}$$

where p_f, p_s denote the pressures on the fluid and solid material surfaces, and $\mathbf{n}$ is the normal vector. These requirements may be combined by employing a weighted residual method. With the approximations:

$$p_s = N_s^i p_{is} \quad , \quad p_f = N_f^j p_{jf} \quad , \tag{14}$$

we have

$$\int N_s^i N_s^j d\Gamma p_{js} = \int N_s^i N_f^j d\Gamma p_{jf} \quad . \tag{16}$$

which may be rewritten as:

$$\mathbf{M}\mathbf{p}_s = \mathbf{r} = \mathbf{L}\mathbf{p}_f \quad . \tag{17}$$

Here $\mathbf{M}$ is a 'consistent mass-matrix', and $\mathbf{L}$ a 'loading matrix'. The solution of this coupled system of equations is obtained iteratively in the now familiar way:

$$\mathbf{M}_l \cdot (\mathbf{p}_s^{i+1} - \mathbf{p}_s^i) = \mathbf{r} - \mathbf{M} \cdot \mathbf{p}_s^i \quad , \tag{18}$$

where $\mathbf{M}_l$ is the 'lumped mass matrix'. Typically, three iterations are sufficient to achieve an accurate result. One can also show that Eqn.(16) is equivalent to the least-squares minimization of

$$I = \int [p_s - p_f]^2 \, d\Gamma, \tag{19}$$

We remark that the weighted residual method is conservative in the sense of Eqn.(13). The sum of all shape-functions at any given point is unity ($\sum_i N_s^i(\mathbf{x}) = 1$), and therefore:

$$\begin{aligned} \int p_s d\Gamma &= \int N_s^j d\Gamma p_{js} = \int \sum_i N_s^i N_s^j d\Gamma p_{js} \\ &= \sum_i \int N_s^i N_s^j d\Gamma p_{js} = \sum_i \int N_s^i N_f^j d\Gamma p_{jf} \\ &= \int \sum_i N_s^i N_f^j d\Gamma p_{jf} = \int N_f^j d\Gamma p_{jf} = \int p_f d\Gamma \end{aligned} \tag{20}$$

The most problematic part of the weighted residual method is the evaluation of the integrals appearing on the right-hand side of Eqn.(16). When the CFD and CSD surface meshes are not nested, this is a formidable task. We have chosen to use Gaussian quadrature. Two options are possible:

Option 1: Perform a loop over the CSD faces

$$\begin{aligned} r^i &= \int N_s^i N_f^j d\Gamma p_{jf} \\ &= \sum_s A_s \sum_{gp} W_{gp} N_s^i(\mathbf{x}_{gp}) p_f(\mathbf{x}_{gp}) \quad . \end{aligned} \tag{21}$$

In this case, the shape-functions $N_s^i(\mathbf{x}_{gp})$ at the Gauss-points are known, and the unknown pressure $p_f(\mathbf{x}_{gp}) = N_f^j(\mathbf{x}_{gp}) p_{jf}$ at the Gauss-points is interpolated. This procedure is non-conservative, as the quadrature points of the solid faces may miss some of the fluid surface pressures.

Option 2: Perform a loop over the CFD faces

$$\begin{aligned} r^i &= \int N_s^i N_f^j d\Gamma p_{jf} \\ &= \sum_f A_f \sum_{gp} W_{gp} N_s^i(\mathbf{x}_{gp}) p_f(\mathbf{x}_{gp}) \quad . \end{aligned} \tag{22}$$

In this case, the pressures $p_f(\mathbf{x}_{gp}) = N_f^j(\mathbf{x}_{gp}) p_{jf}$ at the Gauss-points are known, and the unknown shape-functions $N_s^i(\mathbf{x}_{gp})$ have to be interpolated. This procedure is strictly conservative, as the loop is over the CFD faces, making sure that all pressures present are transmitted to the CSD surface. On the other hand, this procedure can be inaccurate: if the fluid faces are substantially larger than the solid faces, some solid points may not receive any fluid pressure contributions at all. Such a situation has been sketched in Figure 9. In order to avoid these inaccuracies, the face-sizes of the solid faces are interpolated to the fluid surface points. If a large discrepancy in size is encountered, the number of Gauss-points for the fluid faces is adaptively increased. This corresponds to an adaptive refinement of the fluid faces to match the size of the solid faces.

Many of the CSD codes accept only a constant load over each of the surface faces. If this is the case, the elaborate procedure just described may be simplified by taking a constant weighting function P_s^i for each one of the solid faces. The weighted residual statement, for the second, conservative option, then becomes:

$$A_{is} p_{is} = \sum_f A_f \sum_{gp} W_{gp} P_s^i(\mathbf{x}_{gp}) p_f(\mathbf{x}_{gp}) \quad . \tag{23}$$

Given that the polynomial order of the integrals has been reduced from quadratic to linear, only one Gauss-point is required for exact integration. Alternatively, one can compute pressures as before assuming a linear or bilinear pressure disctribution, and then average over the nodes of a surface face. This will tend to spread the loads more evenly over the structural surface faces, avoiding 'ringing'.

9. EXAMPLE RUNS

The main area of applications envisioned for the present fluid-structure algorithm is the interaction of shocks with structures. Therefore, a series of structures were hit by shocks of varying strengths in order to see the ensuing effect.

9.1 Shock-Cylinder Interaction: For this case, we assume that a cylindrical shell of constant thickness and elastic/ideal plastic material (mild steel) is clamped to the soil. A strong shock impacts on the shell, leading to buckling and subsequent collapse. The initial conditions for the fluid, as well as the geometrical and material parameters for the solid are given in Figure 10a. The CFD mesh consisted of approximately 20 Kpts and 100 Ktets. The CSD mesh consisted of approximately 2.3 Kpts and 2.3 Kqshells. Both the CFD and the CSD grids were not adapted in time. A study was conducted to assess the effect of shell thickness on the collapse of the cylinder. Figures 10b-e show the surface grids for the CFD and CSD domains at different times, as well as the surface pressure (CFD) and surface velocity (CSD). Although the CSD surface appears to consist of triangular elements, these are quadrilateral faces that have been split into triangles along the shortest diagonal. One can readily observe the onset of buckling, as well as the completely different buckling shape for the two different cases. Note also the change in stagnation pressures due to the change in surface geometry. During these runs, the CFD region was remeshed two times globally, and several times locally, with no modification to the surface mesh.

9.2 Shock-Shelter Interaction: For this case, a closed semicylindrical shelter was assumed. The shelter was considered to be reinforced at the outer edges, the rim, as well as the middle. As before, an elastic, ideally plastic material was used for the shell. A strong shock impacts on the shelter, leading to severe deformations. The CFD mesh, which was adapted every 7 timesteps, varied between approximately 15 Kpts and 150 Kpts (80 Ktets and 800 Ktets). The CSD mesh, which was not adapted in time, consisted of approximately 1.25 Kpts and 1.25 Kqshells. Figures 11a-c show the surface grids and pressures of the shelter at three time instants during the run. Note the grid adaptation for the CFD domain, as well as the onset of severe deformation in the unreinforced regions due to shock impact.

9.3 Four-Room Experiment: For this case, a test section consisting of four rooms was selected. In room 1 (see Figure 12a) an explosion takes place. The CFD mesh consisted of approximately 260 Kpts and 1.3 Mtets. Two CSD models were employed. The first consisted of approximately 50 Kbricks that were generated by splitting a coarser mesh of tetrahedra. This mesh had no reinforcement bars (re-bars) in the model. Rather, the reinforced concrete was treated as a single material with properly averaged elasto-plastic parameters. The second model consisted of approximately 50 Kbricks that had regular brick-shape. For the walls, approriate reinforced concrete and re-bar models were specified. The different grids are shown in Figures 12b,c. Figures 12d,e show the surface pressures for the CFD region, as well as the absolute value of the surface velocities for the CSD region at different times during the run. Note that in one instance, a large velocity is exhibited by the structure for the wall separating the lower rooms, even though the shock waves have not yet reached this section of the domain. This is because the wall acts as a stiff beam, producing a rotational motion around the line where the four inner walls cross. Thus, a positive velocity in the upper wall results in a negative velocity for the lower walls. Figure 12f shows the comparison to experimental measurements for the pressures at four locations on the walls of the first room. Note the excellent agreement between the predictions and the measured data. Figure 12g shows the comparison of the displacements predicted for the two different CSD models with experimental measurements at a point on the upper interior wall. Observe that the experimental data exhibits a linear drift. This phenomenon is commonly observed for accelerometers. Taking into account this drift, one can see that the correlation between the experiment and the predictions is quite good. One can also see that the more detailed model, which included re-bar models, gave a better correlation with the experiment.

9.4 Truck: This last case shows some preliminary results for a realistic shock-object interaction case. The objective in describing this case is not to show detailed comparisons with experiments. A separate report is being prepared for this purpose. Rather, the aim is to show what is possible once a general set of interpolation and projection techniques is combined with interactive pre-processors and production CSD and CFD codes. The geometry considered is that of a 5-ton Army truck subjected to a strong shock. Some of the geometric data was obtained in Auto-Cad format. Simplifying assumptions were made for the engine, transmission and transmission shafts: all of these were simulated as rigid solids. The springs and tires were simulated with eight-noded brick elements of appropriate material behaviour. The complete CAD model for the structure, which was obtained in a week, consisted of 5,928 points, 3,000 lines, and 1,386 surfaces, and is shown in Figure 13a. Once the CAD model for the structure was obtained, the CAD model for the surface was obtained in a single step by invoking an 'invert and unwrap' option in the pre-processor used. This option removes all the inte-

rior CSD CAD data, leaving only the wetted surface data for further use, and unwraps doubly loaded surfaces (e.g. shells). The CAD model for the fluid consisted of 6,306 points, 3,718 lines, and 1,604 surfaces. The FEM models totalled 1 Kbeams, 50 Kqshells, 50 Kbricks and 22 materials for the structure, and 200 Kpoints and 1 Mtets for the fluid. The surface discretizations of these grids are shown in Figure 13b. Figures 13c-d show the results for two different times during the run.

10. CONCLUSIONS AND OUTLOOK

A fluid/structure interaction algorithm based on the loose coupling of production CFD and CSD codes has been described. The algorithm allows a cost-effective re-use of existing software, with a minimum amount of alterations required to account for the interaction of the different media. Several example runs using **FEFLO96** as the CFD code, and **DYNA3D** as the CSD code, demonstrate the effectiveness of the proposed methodology.

Future developments will center on:

- Treatment of reduced models, or models with incompatible dimensionalities;
- Improved reliability for complex geometries undergoing severe deformations, especially when contact is present;
- Further improvements to handle not only accurate load conserving projection, but also work conserving projection;
- Extensions to Navier-Stokes problems for the CFD codes;
- Inclusion of implicit CSD codes, such as NASTRAN, ANSYS, or NIKE-3D to treat steady-state or low frequency problems;
- Extensions to other multidisciplinary problems, including thermal and/or electromagnetic loads.

11. ACKNOWLEDGMENTS

This work was partially supported by DNA and AFOSR, with Drs. Mike Giltrud and Leonidas Sakell as the technical monitors. The authors would also like to thank CRAY Research for its support on the form of many hours on advanced C-90 supercomputers.

12. REFERENCES

[1] CSD codes such as NASTRAN, ANSYS, ABAQUS, MARC, NISA, ADINA, DYNA3D, PAM-CRASH, etc., CFD codes such as FLUENT, FIDAP, STAR-CD, RAMPANT, FEFLO, PAM-FLOW, etc.

[2] O.C. Zienkiewicz and R. Taylor - *The Finite Element Method*; McGraw Hill (1988).

[3] G.C. Everstine and F.M. Henderson - Coupled Finite Element/Boundary Element Approach for Fluid-Structure Interaction; *J. Acoust. Soc. Am.* 87, 5, 1938-1947 (1990).

[4] G.C. Everstine - Prediction of Low Frequency Vibrational Frequencies of Submerged Structures; *J. Vibrations and Acoustics* 113, (1991).

[5] P.S. Jackson and G.W. Christie - Numerical Analysis of Three-Dimensional Elastic Membrane Wings; *AIAA J.* 25, 5, 676-682, (1987).

[6] J.T. Batina, R.M. Bennet, D.A. Seidel, H.J. Cunningham and S.R. Bland - Recent Advances in Transonic Computational Aeroelasticity; *Comp. Struct.* **30**, No.1/2, 29-37, (1988).

[7] J. Alonso, L. Martinelli and A. Jameson - Multigrid Unsteady Navier-Stokes Calculations with Aeroelastic Applications; *AIAA*-95-0048 (1995).

[8] G.P. Guruswamy - Unsteady Aerodynamic and Aerolastic Calculations for Wings Using Euler Equations; *AIAA J.* 28, 3, 461-469 (1990).

[9] R.D. Rausch, J.T. Batina and H.T.Y. Yang - Three-Dimensional Time-Marching Aerolastic Analyses Using an Unstructured-Grid Euler Method; *AIAA J.* 31, 9, 1626-1633 (1993).

[10] A.H. Boschitsch and T.R. Quackenbush - High Accuracy Computations of Fluid-Structure Interaction in Transonic Cascades; *AIAA*-93-0485 (1993).

[11] F.F. Felker - Direct Solution of Two-Dimensional Navier-Stokes Equations for Static Aeroelasticity Problems; *AIAA J.* 31, 1, 148-153 (1993).

[12] G.P. Guruswamy and C. Byun - Fluid-Structural Interactions Using Navier-Stokes Flow Equations Coupled with Shell Finite Element Structures; *AIAA*-93-3087 (1993).

[13] J. Donea - An Arbitrary Lagrangian-Eulerian Finite Element Method for Transient Dynamic Fluid-Structure Interactions; *Comp. Meth. Appl. Mech. Eng.* 33, 689-723 (1982).

[14] B. van Leer - Towards the Ultimate Conservative Scheme. II. Monotonicity and Conservation Combined in a Second Order Scheme; *J.Comp.Phys.* 14, 361-370 (1974).

[15] P.L. Roe - Approximate Riemann Solvers, Parameter Vectors and Difference Schemes; *J.Comp.Phys.* 43, 357-372 (1981).

[16] A. Jameson - Artificial Diffusion, Upwind Biasing, Limiters and Their Effect on Accuracy and Multigrid Convergence in Transonic and Hypersonic Flows; *AIAA*-93-3359 (1993).

[17] R. Löhner, K. Morgan, J. Peraire and M. Vahdati - Finite Element Flux-Corrected Transport

(FEM-FCT) for the Euler and Navier-Stokes Equations; *Int. J. Num. Meth. Fluids* 7, 1093-1109 (1987).

[18] R. Löhner and J. Ambrosiano - A Vectorized Particle Tracer for Unstructured Grids; *J. Comp. Phys.* 91, 1, 22-31 (1990).

[19] R. Löhner and J.D. Baum - Adaptive H-Refinement on 3-D Unstructured Grids for Transient Problems; *Int. J. Num. Meth. Fluids* 14, 1407-1419 (1992).

[20] R. Löhner and P. Parikh - Three-Dimensional Grid Generation by the Advancing Front Method; *Int. J. Num. Meth. Fluids* 8, 1135-1149 (1988).

[21] S. Sivier, E. Loth, J.D. Baum and R. Löhner - Unstructured Adaptive Remeshing Finite Element Method for Dusty Shock Flows; *Shock Waves* 4, 15-23 (1994).

[22] R. Löhner - Some Useful Renumbering Strategies for Unstructured Grids; *Int. J. Num. Meth. Eng.* 36, 3259-3270 (1993).

[23] R. Löhner, R. Ramamurti and D. Martin - A Parallelizable Load Balancing Algorithm; *AIAA*-93-0061 (1993).

[24] J.D. Baum, H. Luo and R. Löhner - Numerical Simulation of a Blast Inside a Boeing 747; *AIAA*--93-3091 (1993).

[25] H. Luo, J.D. Baum, R. Löhner and J. Cabello - Adaptive Edge-Based Finite Element Schemes for the Euler and Navier-Stokes Equations; *AIAA*-93-0336 (1993).

[26] J.D. Baum, H. Luo and R. Löhner - A New ALE Adaptive Unstructured Methodology for the Simulation of Moving Bodies; *AIAA*-94-0414 (1994).

[27] H. Luo, J.D. Baum and R. Löhner - Edge-Based Finite Element Scheme for the Euler Equations; *AIAA J.* 32, 6, 1183-1190 (1994).

[28] J.D. Baum, H. Luo and R. Löhner - Numerical Simulation of Blast in the World Trade Center; *AIAA*-95-0085 (1995).

[29] R.G. Whirley and J.O. Hallquist - DYNA3D, A Nonlinear Explicit, Three-Dimensional Finite Element Code for Solid and Structural Mechanics - User Manual; UCRL-MA-107254, Rev.1, (1993).

[30] T. Belytschko and Tsay - Explicit Algorithm for Nonlinear Dynamics of Shells; *Comp. Meth. Appl. Mech. Eng.* 43, 251-276 (1984).

[31] T.J.R. Hughes and W.K. Liu - Nonlinear Finite Element Analysis of Shells: Part I. Three Dimensional Shells; *Comp. Meth. Appl. Mech. Eng.* 26, 331-362 (1981).

[32] T. Belytschko and I. Leviathan - Physical Stabilization of the 4-Node Shell Element with One Point Quadrature; *Comp. Meth. Appl. Mech. Eng.* 113, 321-350, (1994).

[33] D.P. Flanagan and T. Belytschko - A Uniform Strain Hexahedron and Quadrilateral with Orthogonal Hourglass Control; *Int. J. Num. Meth. Eng.* 17, 679-706, (1981).

[34] G.L. Goudreau and J.O. Hallquist - Recent Developments in Large-Scale Finite Element Lagrangean Hydrocode Technology; *Comp. Meth. Appl. Mech. Eng.* 33, 725-757 (1982).

[35] C.M. Charman, R.M. Grenier, and R.R. Nickell - Large Deformation Inelastic Analysis of Impact for Shipping Casks; *Comp. Meth. Appl. Mech. Eng.* 33, 759-784, (1982).

[36] R. Löhner - Robust, Vectorized Search Algorithms for Interpolation on Unstructured Grids; *J. Comp. Phys.* 118, 380-387 (1995).

[37] D.N. Knuth - *The Art of Computer Programming* , Vol. 3; Addison-Wesley (1973).

[38] R. Sedgewick - *Algorithms* ; Addison-Wesley (1983).

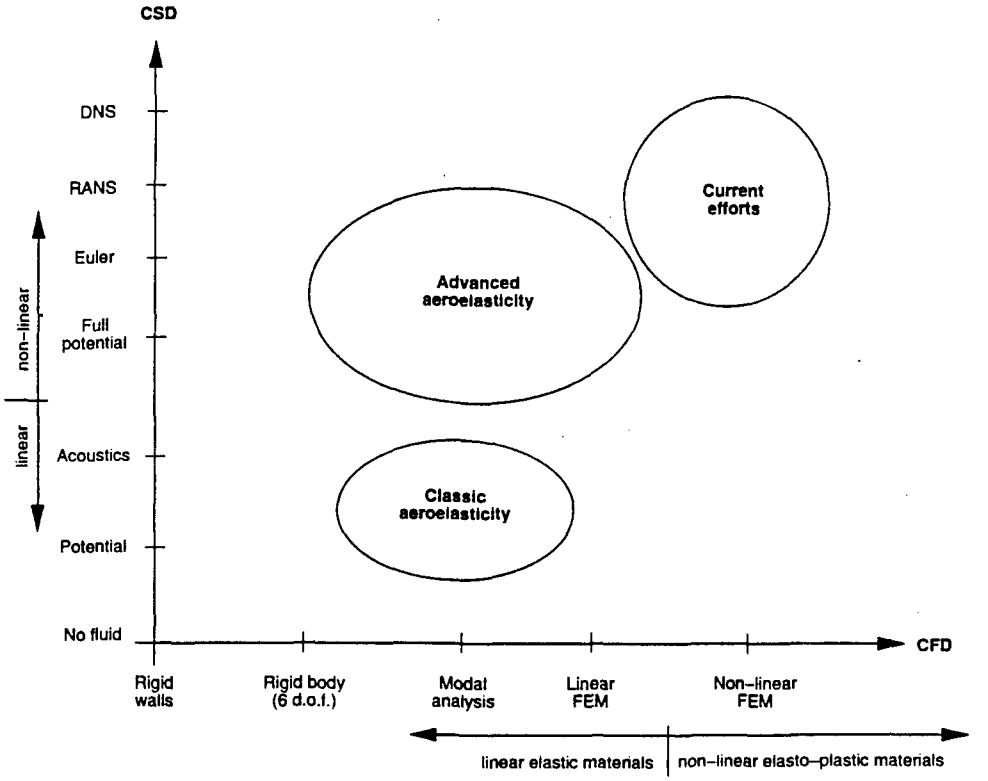


Figure 1: Levels of Physical Complexity for CFD and CSD

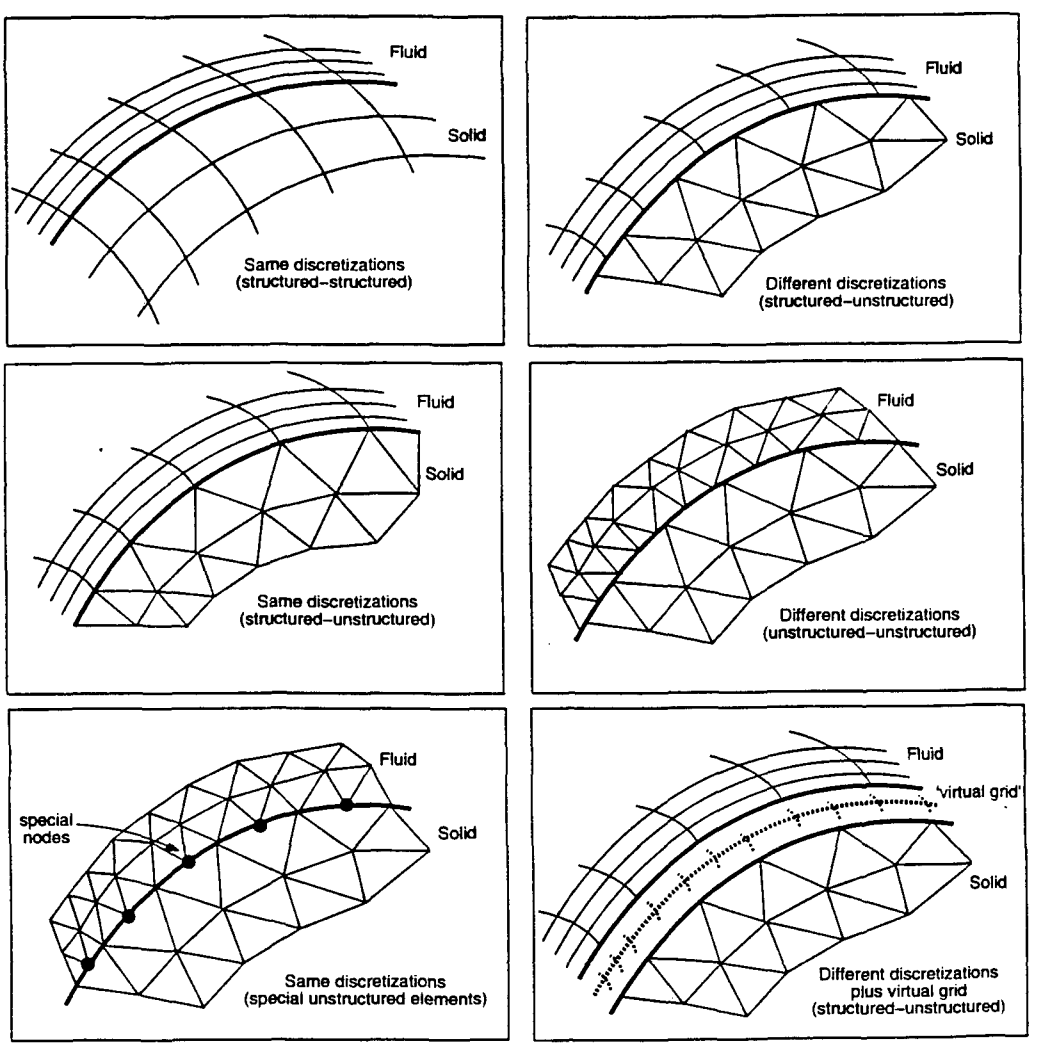


Figure 2: Surface Discretizations for CFD and CSD

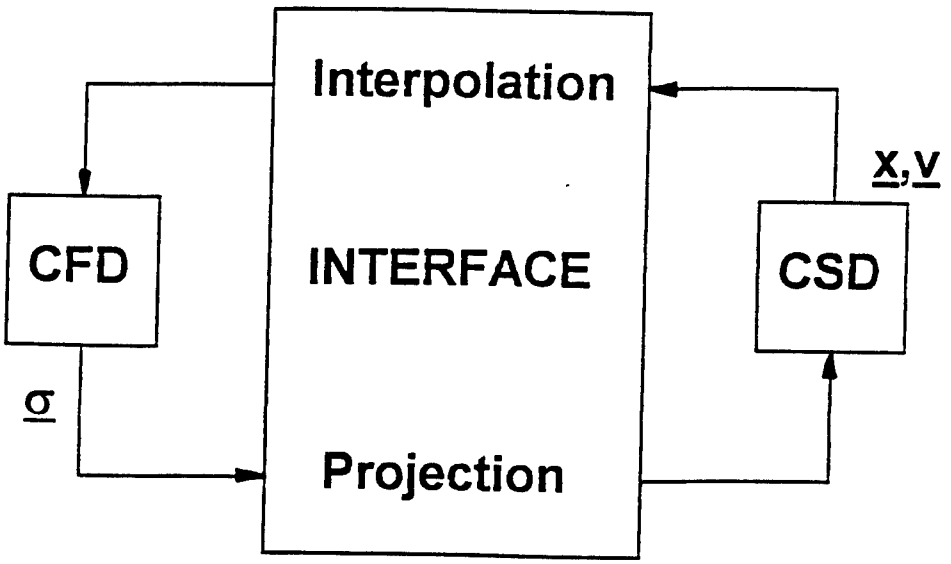
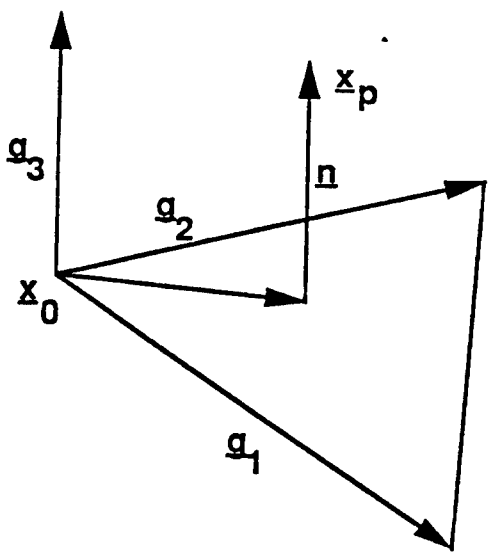


Figure 3: Loose Coupling Algorithm



$$x_p = x_0 + \alpha^l g_l$$

$$d_n = | \alpha^3 g_3 |$$

Figure 4: Surface to Surface Grid Interpolation

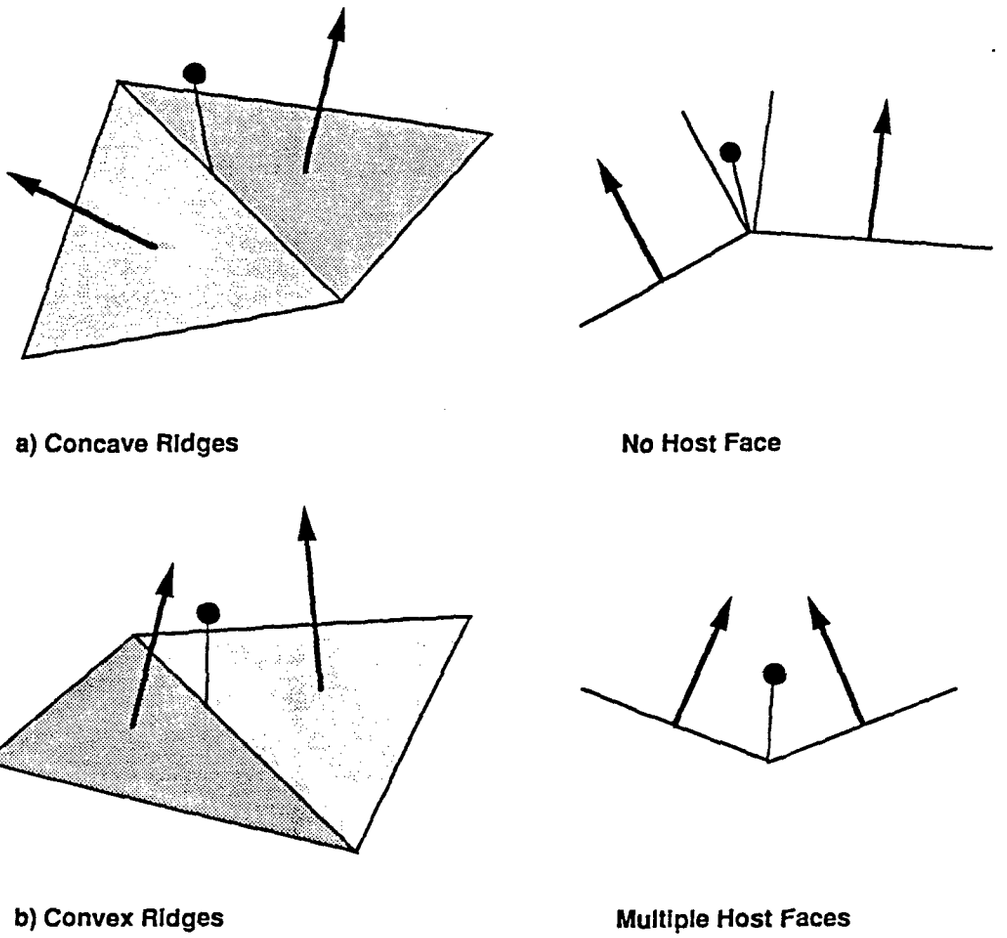


Figure 5: Problems When Searching for Host Faces

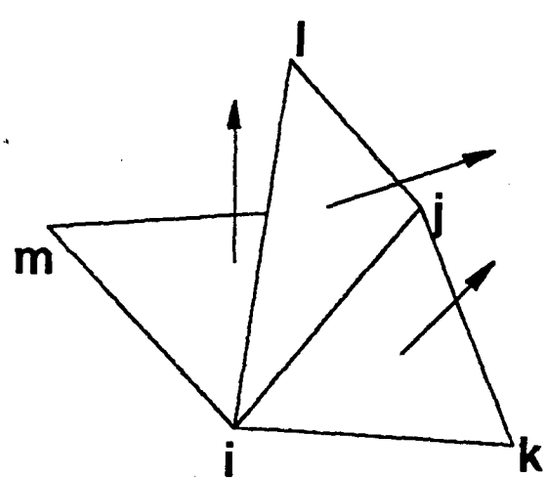


Figure 6: Most Visible Neighbour Face

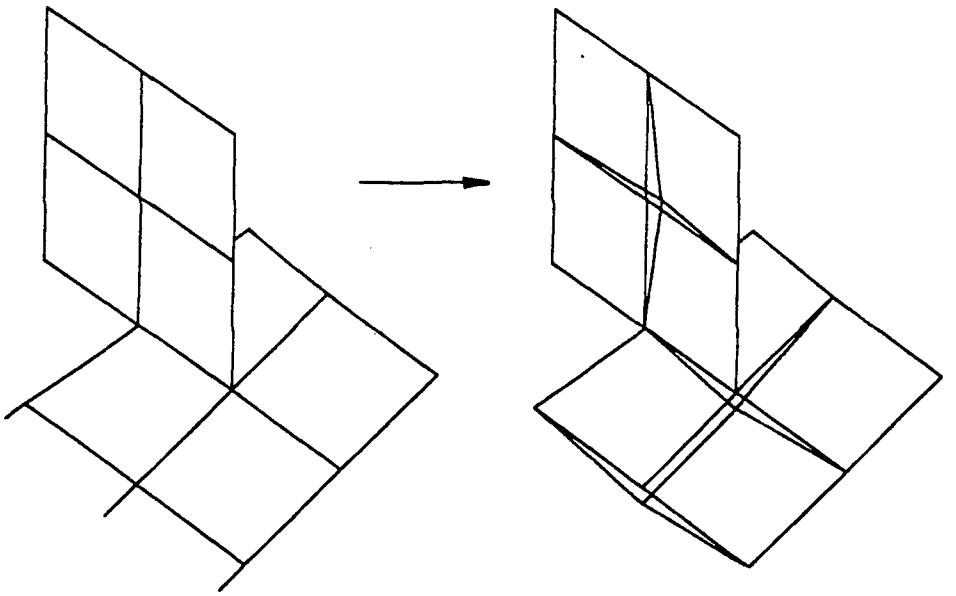
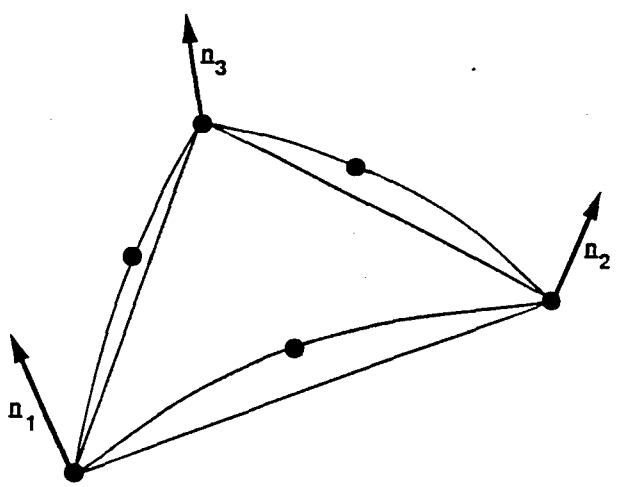
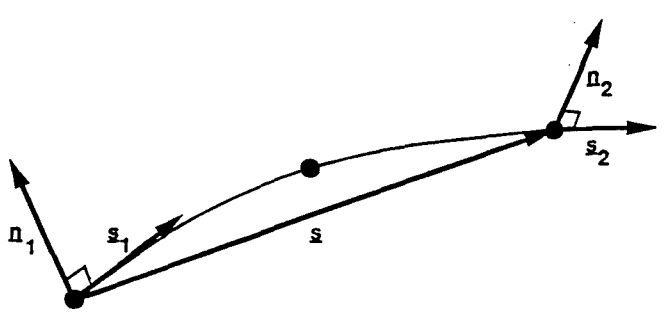


Figure 7: Unwrapping Doubly Defined Faces



Surface Face With Normals



Surface Edge

Figure 8: Quadratic Surface Reconstruction With Edge-Wise Hermite Polynomials

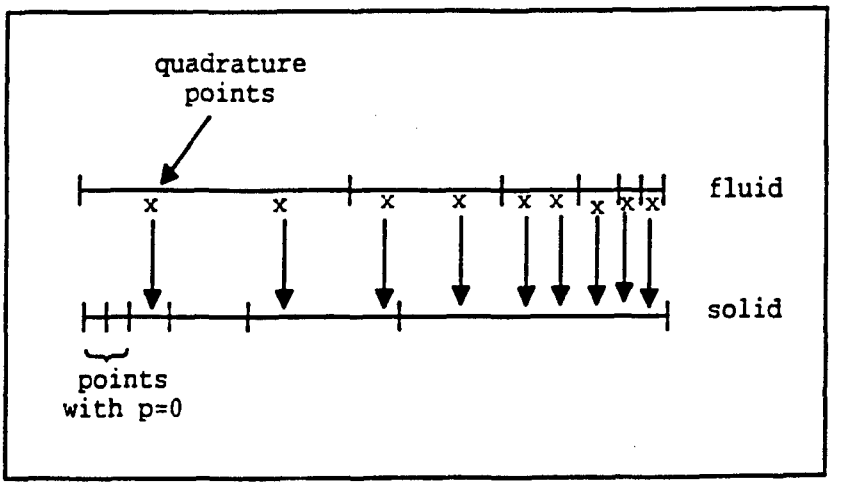


Figure 9: Quadrature Points Introduced on Fluid Faces

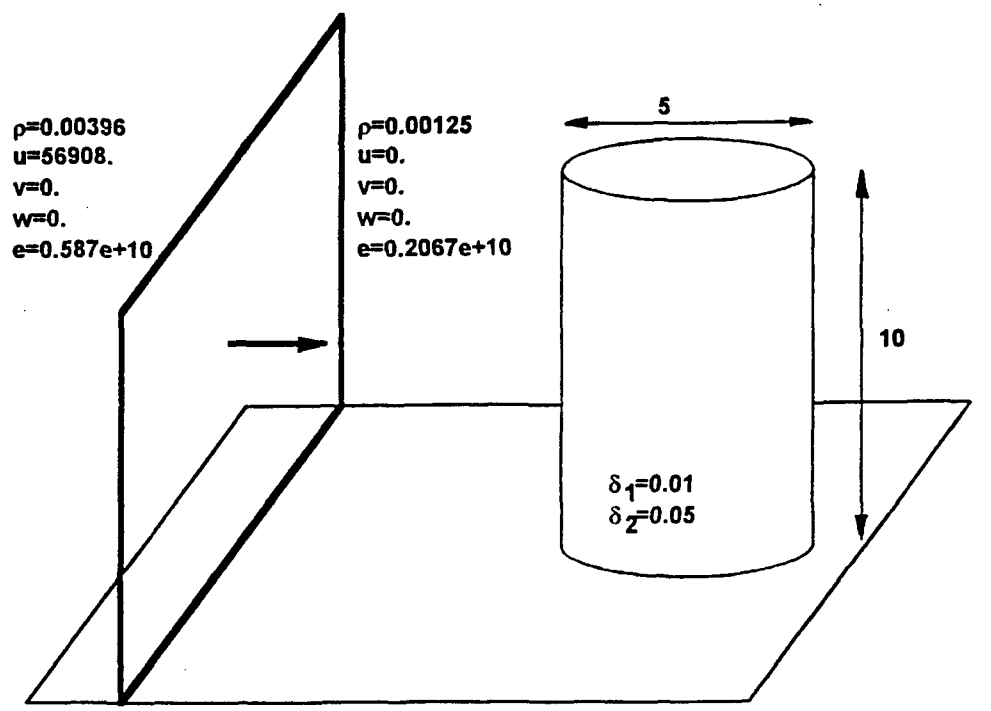
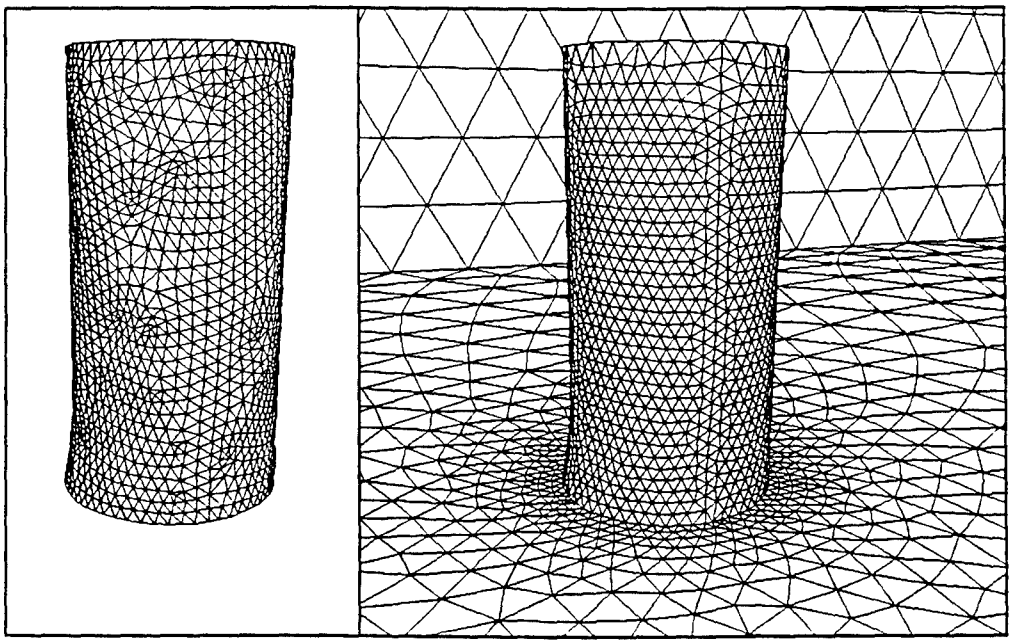


Figure 10: Shock-Cylinder Interaction

a) Problem Definition

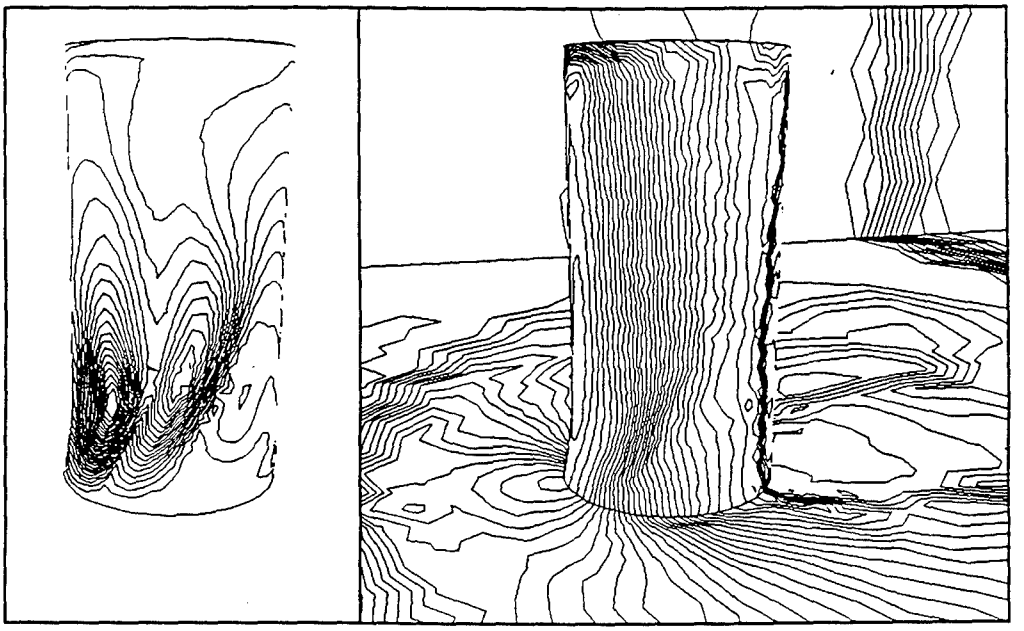
SHOCK–CYLINDER INTERACTION (T=0.39E–03)



Dyna3d Surface Mesh

FEFLO96 Surface Mesh

SHOCK–CYLINDER INTERACTION (T=0.39E–03)

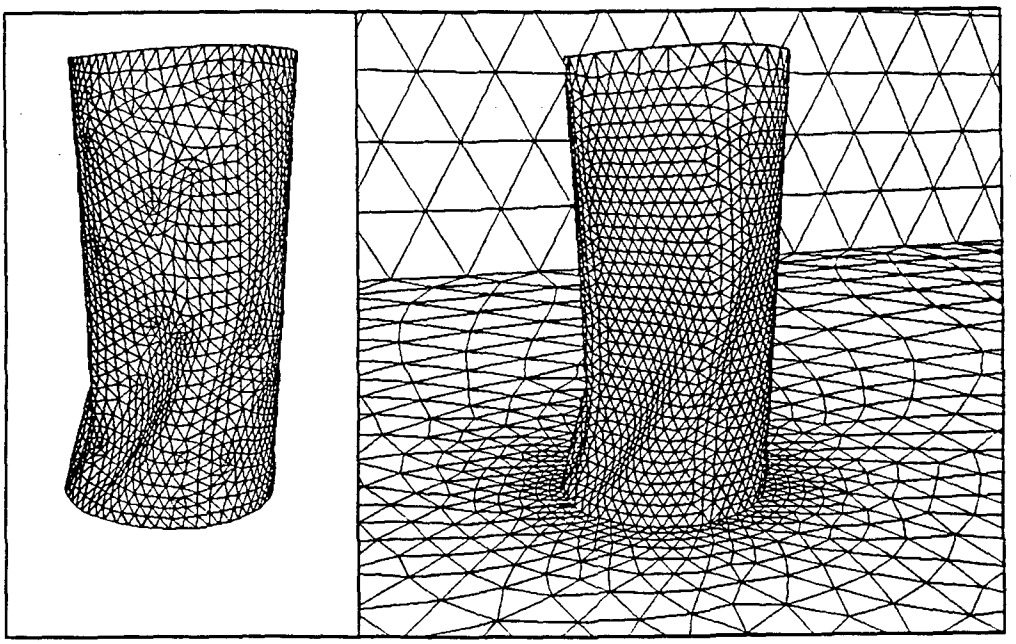


Dyna3d Abs. Velocity

FEFLO96 Pressure

b) Thick Shell: T=0.39e-3

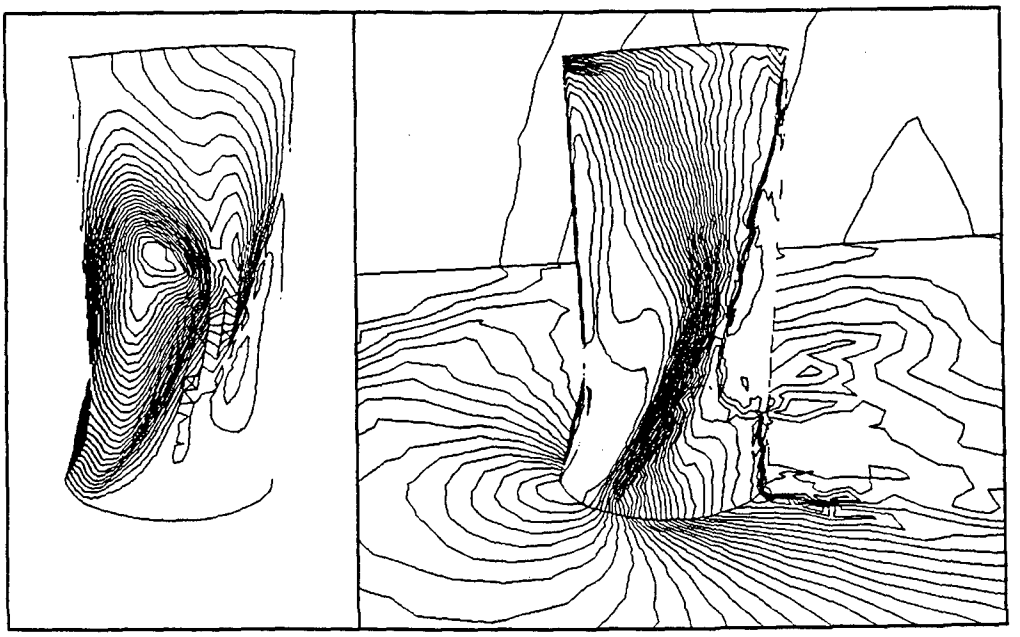
SHOCK–CYLINDER INTERACTION (T=0.79E–03)



Dyna3d Surface Mesh

FEFLO96 Surface Mesh

SHOCK–CYLINDER INTERACTION (T=0.79E–03)

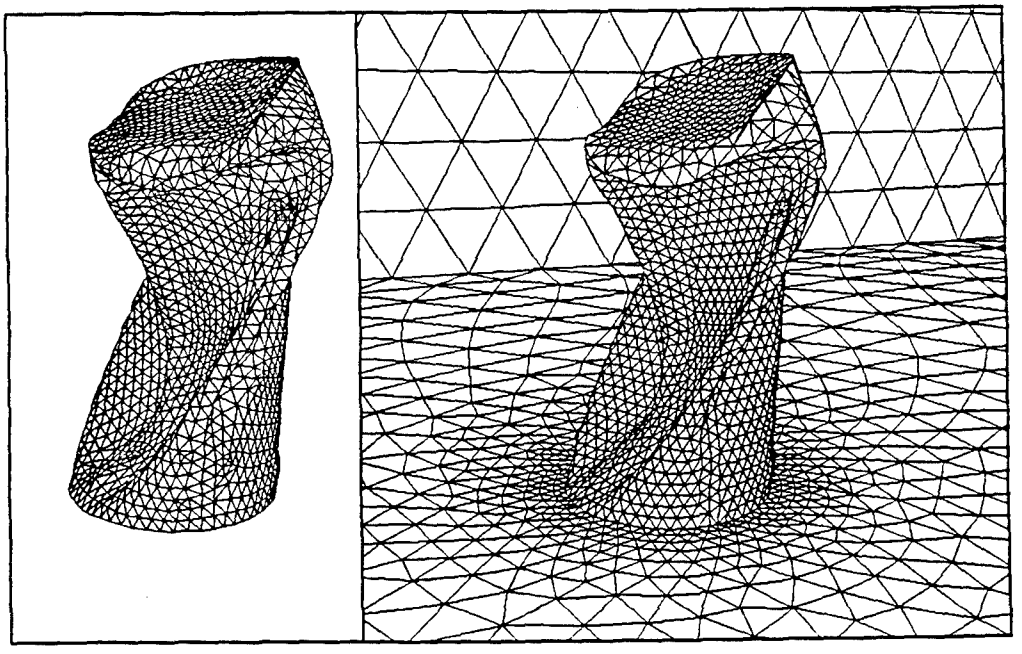


Dyna3d Abs. Velocity

FEFLO96 Pressure

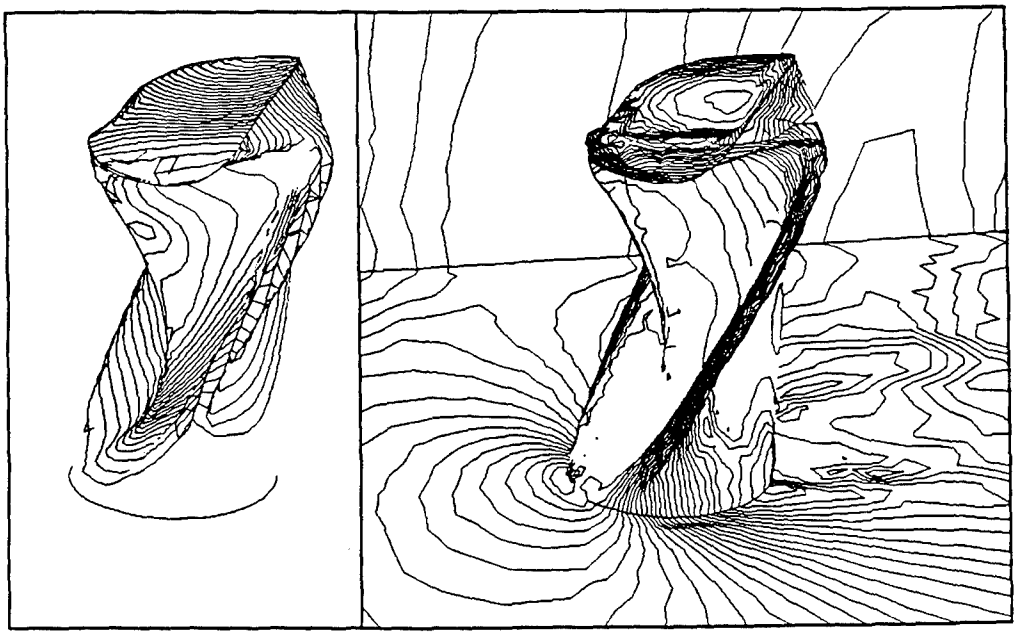
c) Thick Shell: T=0.79e-3

SHOCK–CYLINDER INTERACTION (T=1.41E–03)



Dyna3d Surface Mesh FEFLO96 Surface Mesh

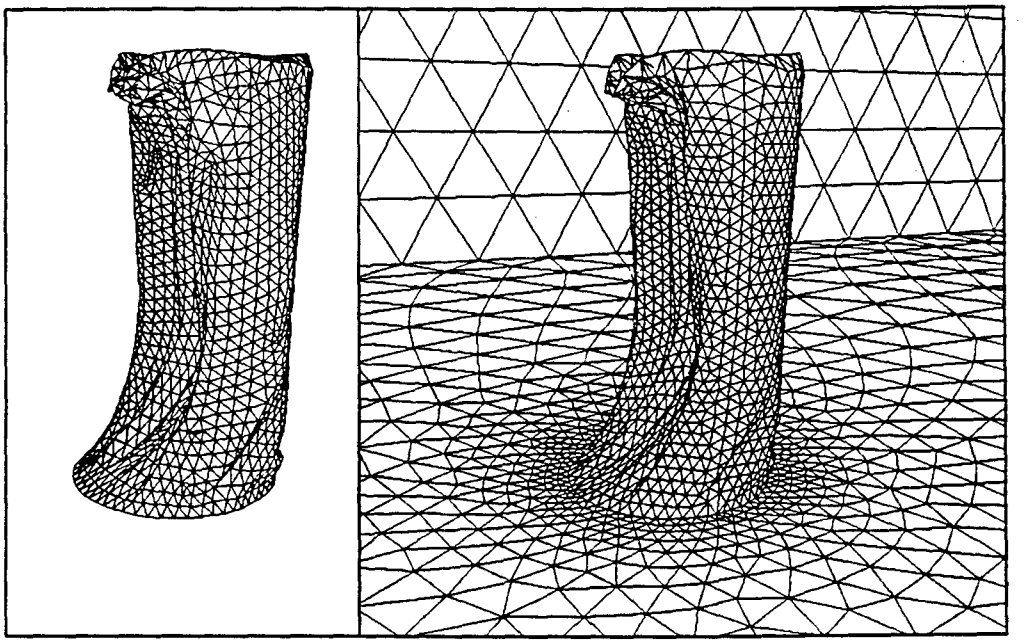
SHOCK–CYLINDER INTERACTION (T=1.41E–03)



Dyna3d Abs. Velocity FEFLO96 Pressure

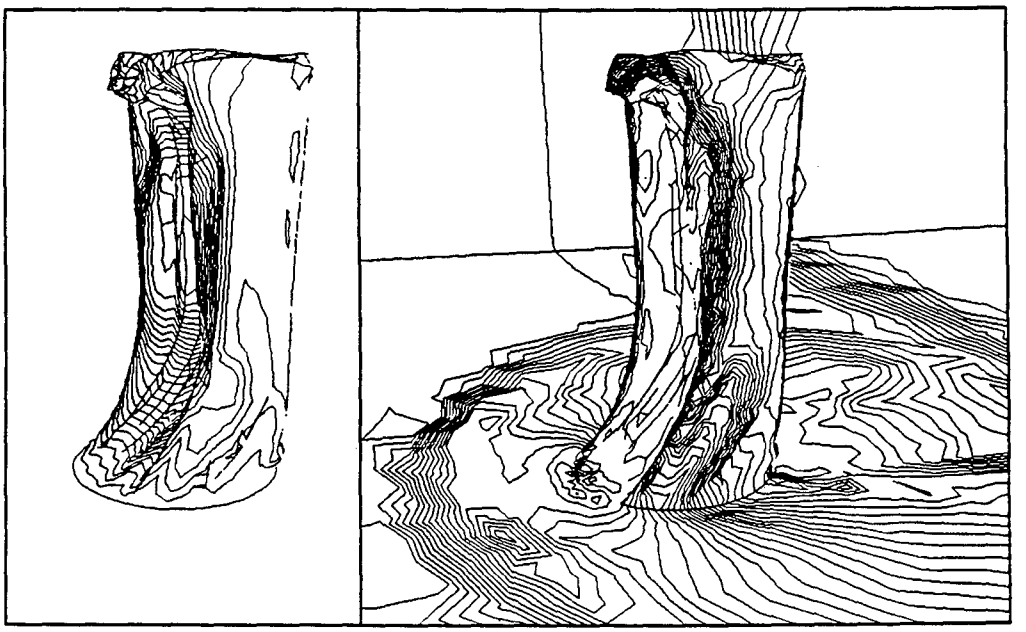
d) Thick Shell: T=1.41e-3

SHOCK–CYLINDER INTERACTION (T=0.24E–03)



Dyna3d Surface Mesh FEFLO96 Surface Mesh

SHOCK–CYLINDER INTERACTION (T=0.24E–03)

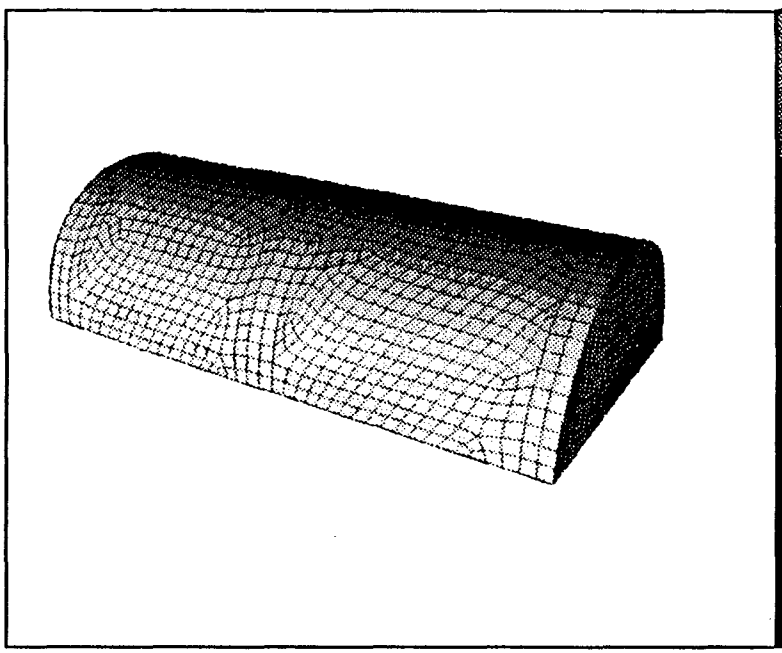


Dyna3d Abs. Velocity FEFLO96 Pressure

e) Thin Shell: T=0.24e-3

SHOCK INTERACTION WITH A SHELTER

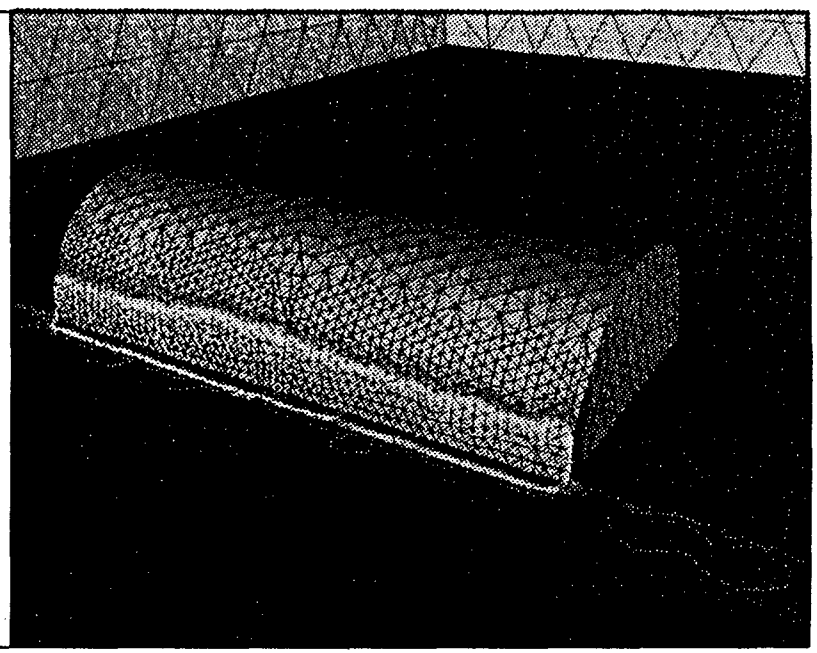
DYNA3D Surface Mesh



NPOIN=1,953
NQUAD=1,890

100 Steps, T=1.71e-3

FEFLO96 Surface Mesh and Pressures



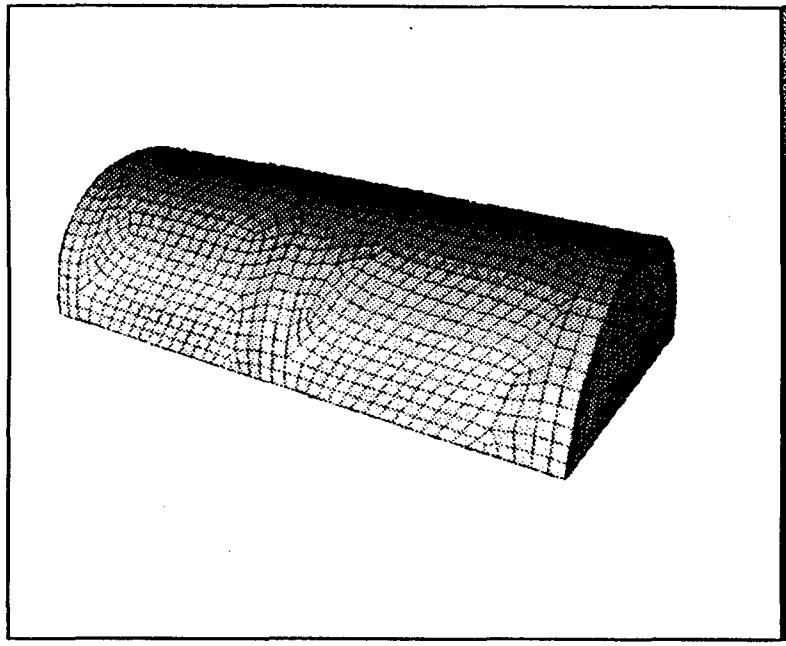
NPOIN=88,000
NELEM=500,000

a) T=1.71e-3: Surface Grids and Pressure

Figure 11: Shock-Shelter Interaction

SHOCK INTERACTION WITH A SHELTER

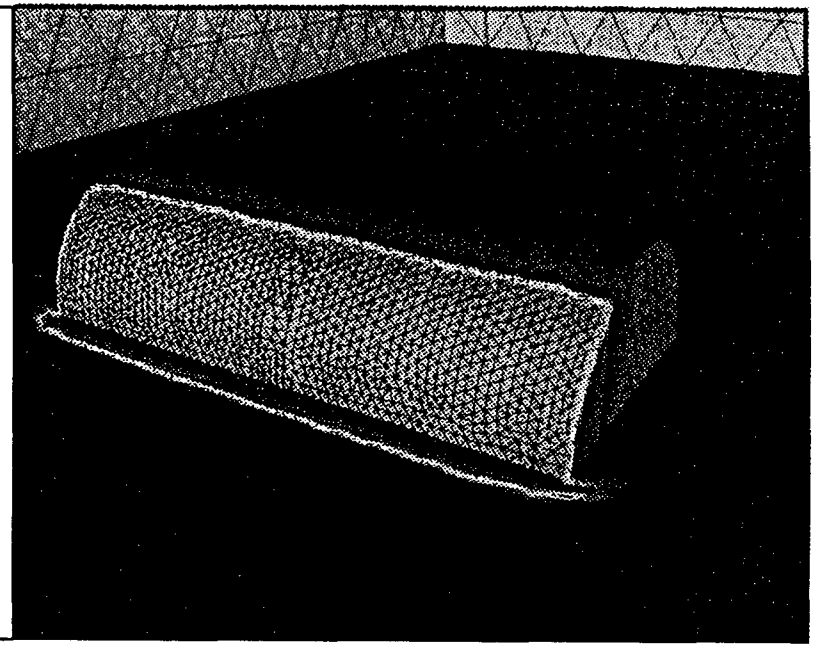
DYNA3D Surface Mesh



NPOIN=1,953
NQUAD=1,890

200 Steps, T=3.17e–3

FEFLO96 Surface Mesh and Pressures

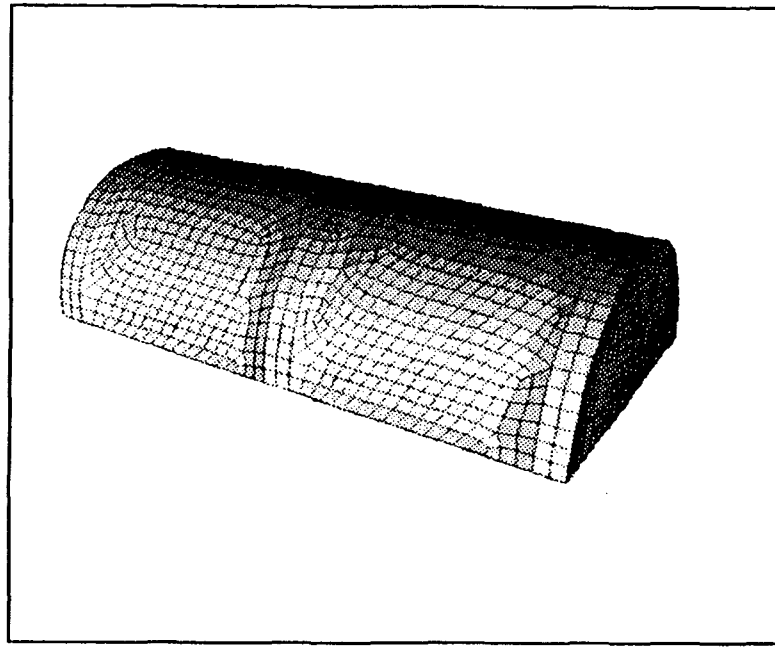


NPOIN=107,000
NELEM=610,000

b) T=3.17e-3: Surface Grids and Pressure

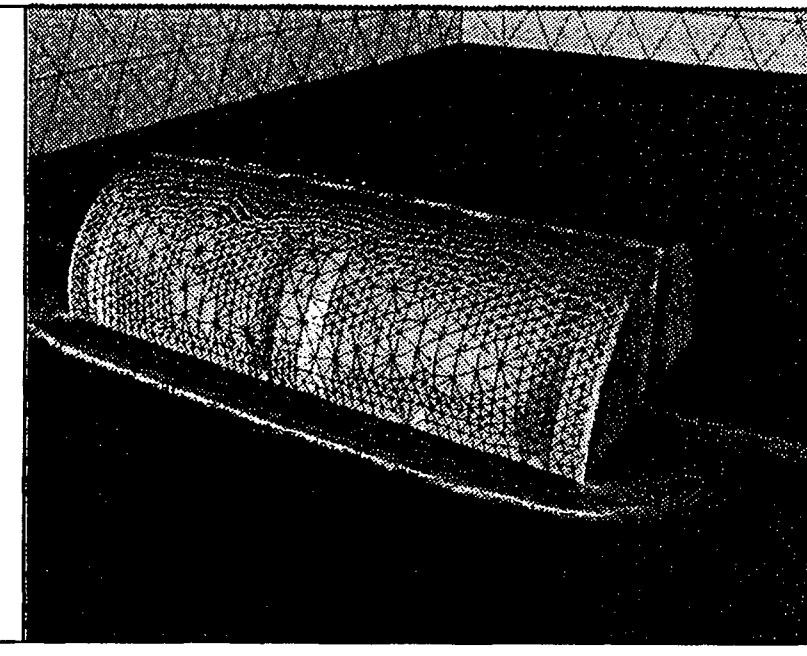
SHOCK INTERACTION WITH A SHELTER

DYNA3D Surface Mesh



NPOIN=1,953
NQUAD=1,890

FEFLO96 Surface Mesh and Pressures



NPOIN=127,000
NELEM=728,000

300 Steps, T=4.67e–3

c) T=4.67e-3: Surface Grids and Pressure

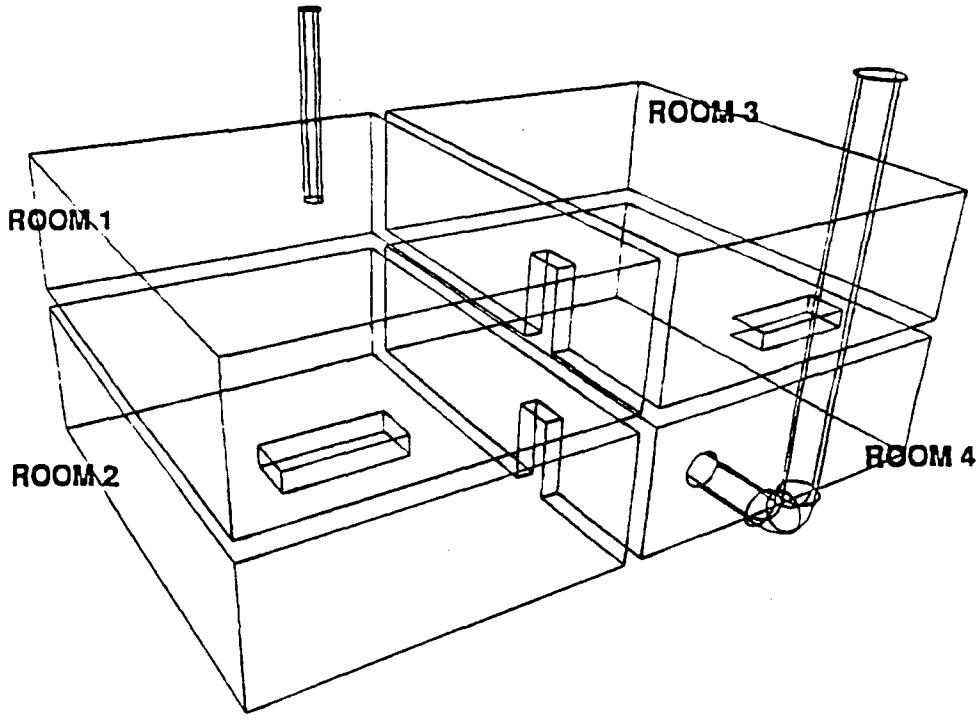
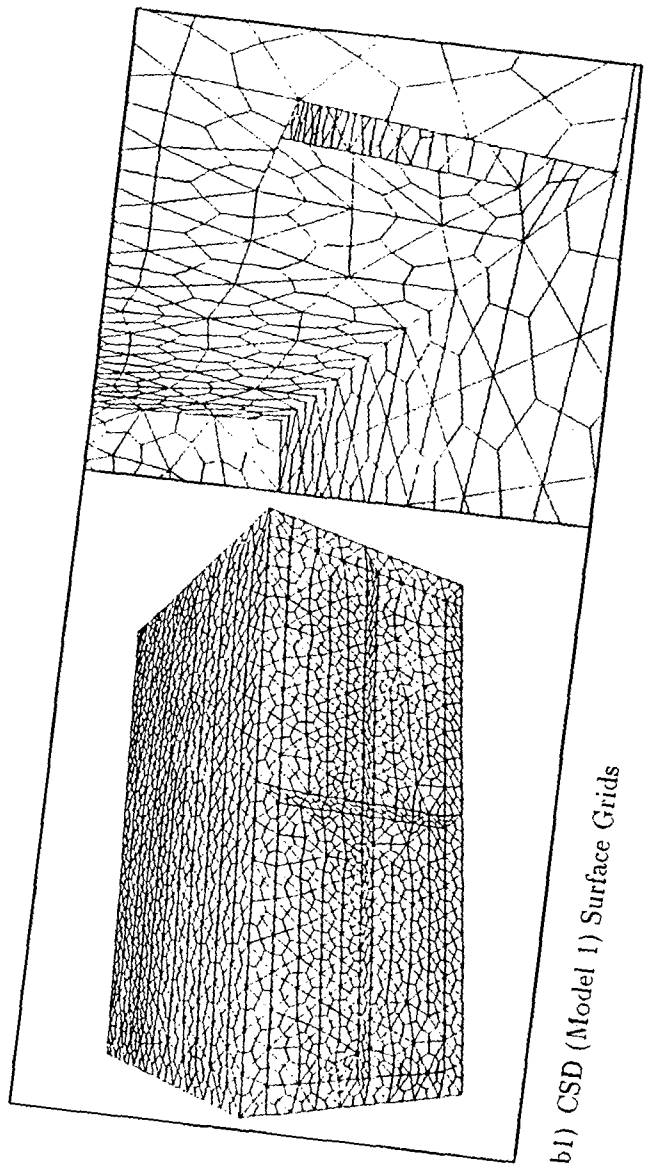
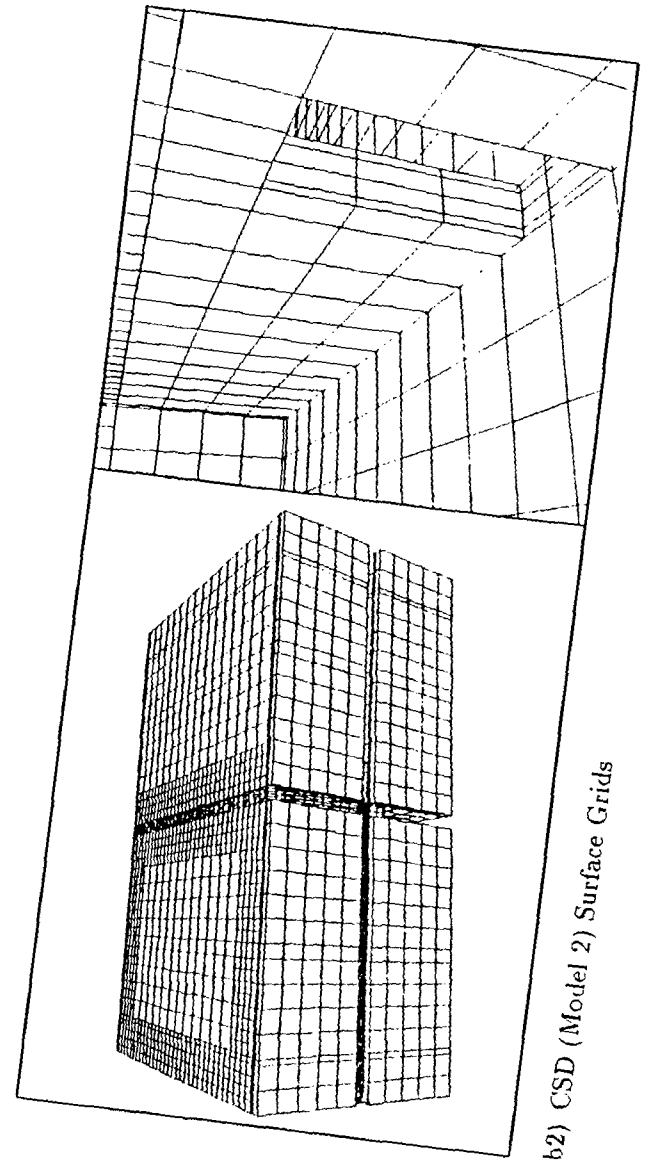


Figure 12: Four-Room Experiment

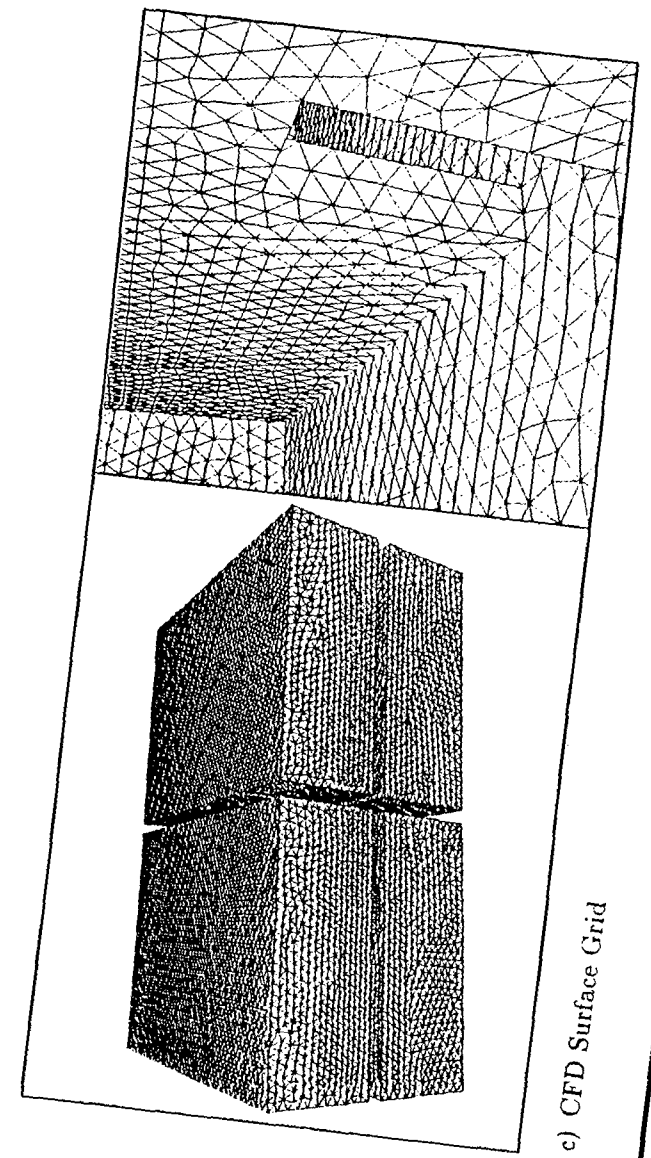
a) Geometry Definition



b1) CSD (Model 1) Surface Grids



b2) CSD (Model 2) Surface Grids

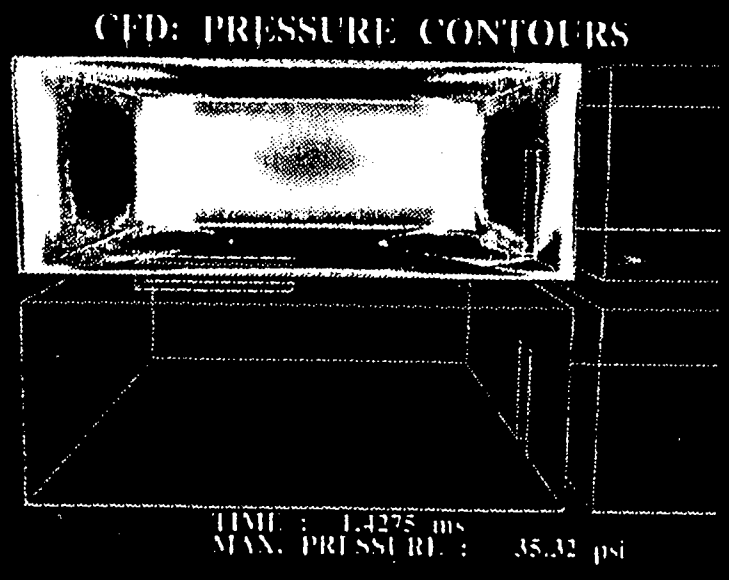


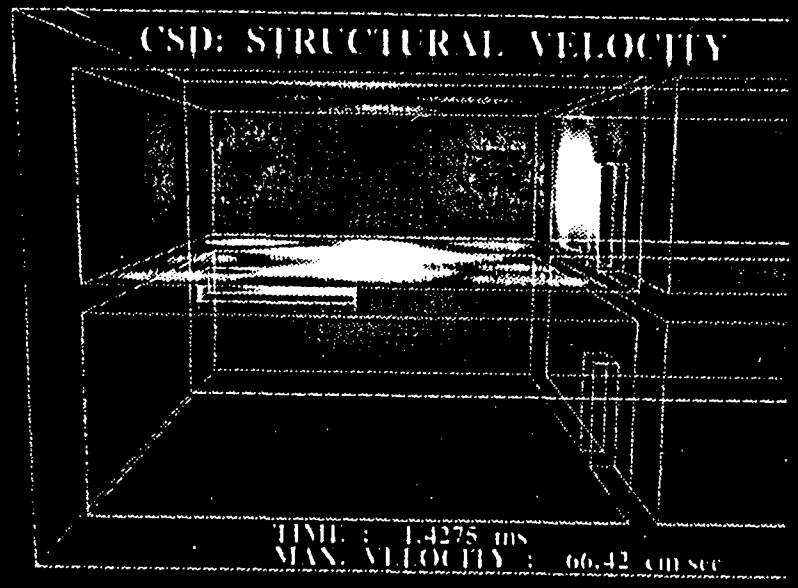
c) CFD Surface Grid

CFD – CSD INTEGRATION

FEFLO96

DYNA3D



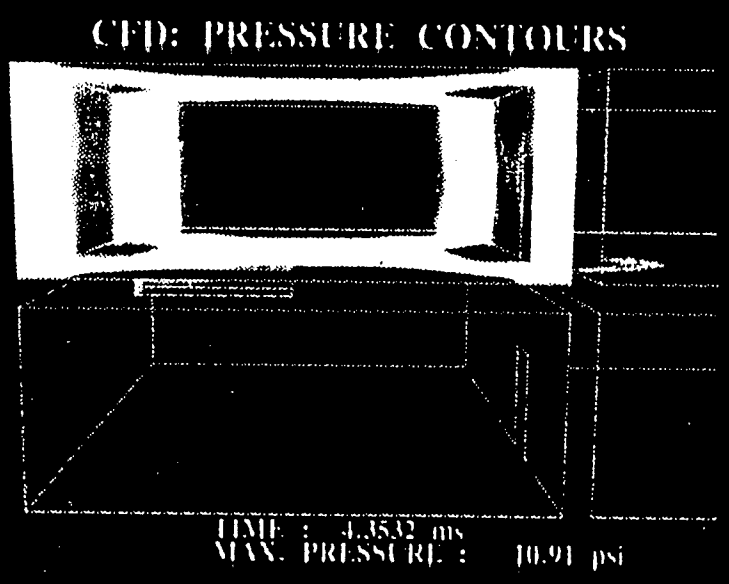


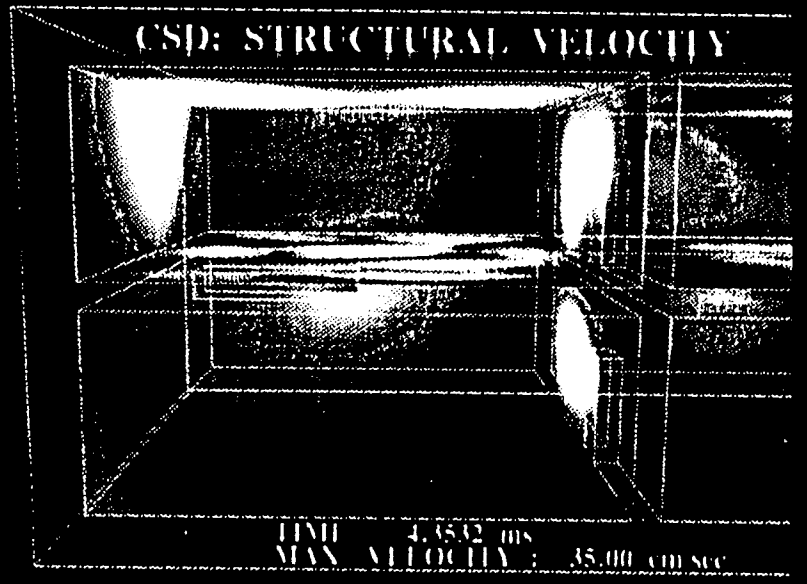
SAIC / GMU / GA

CFD – CSD INTEGRATION

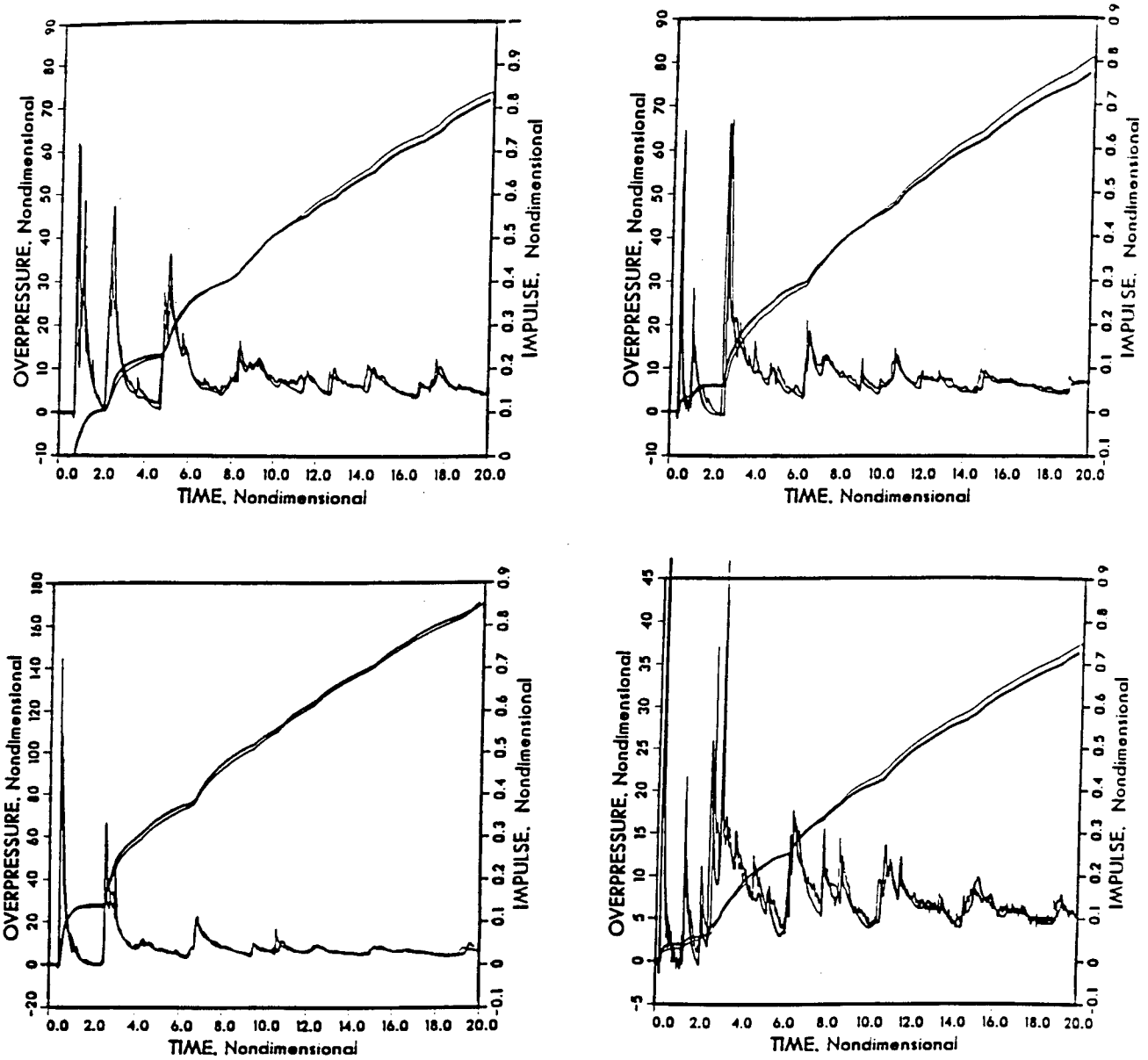
FEFLO96

DYNA3D

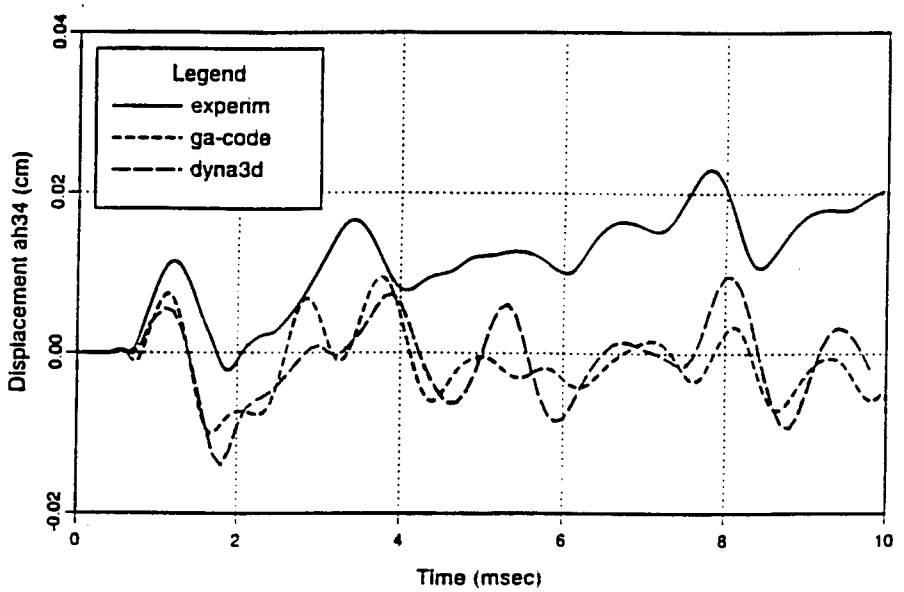




SAIC / GMU / GA

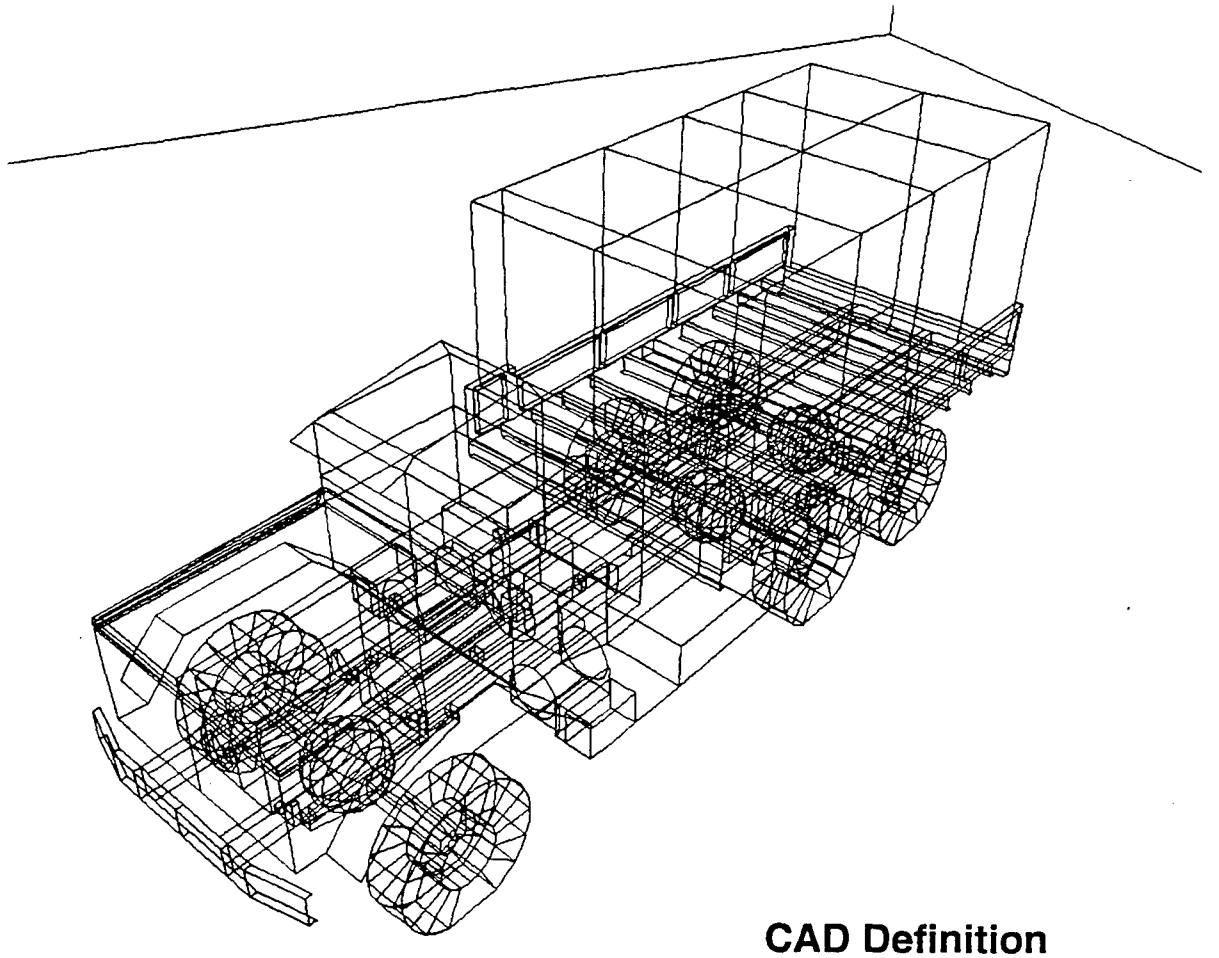


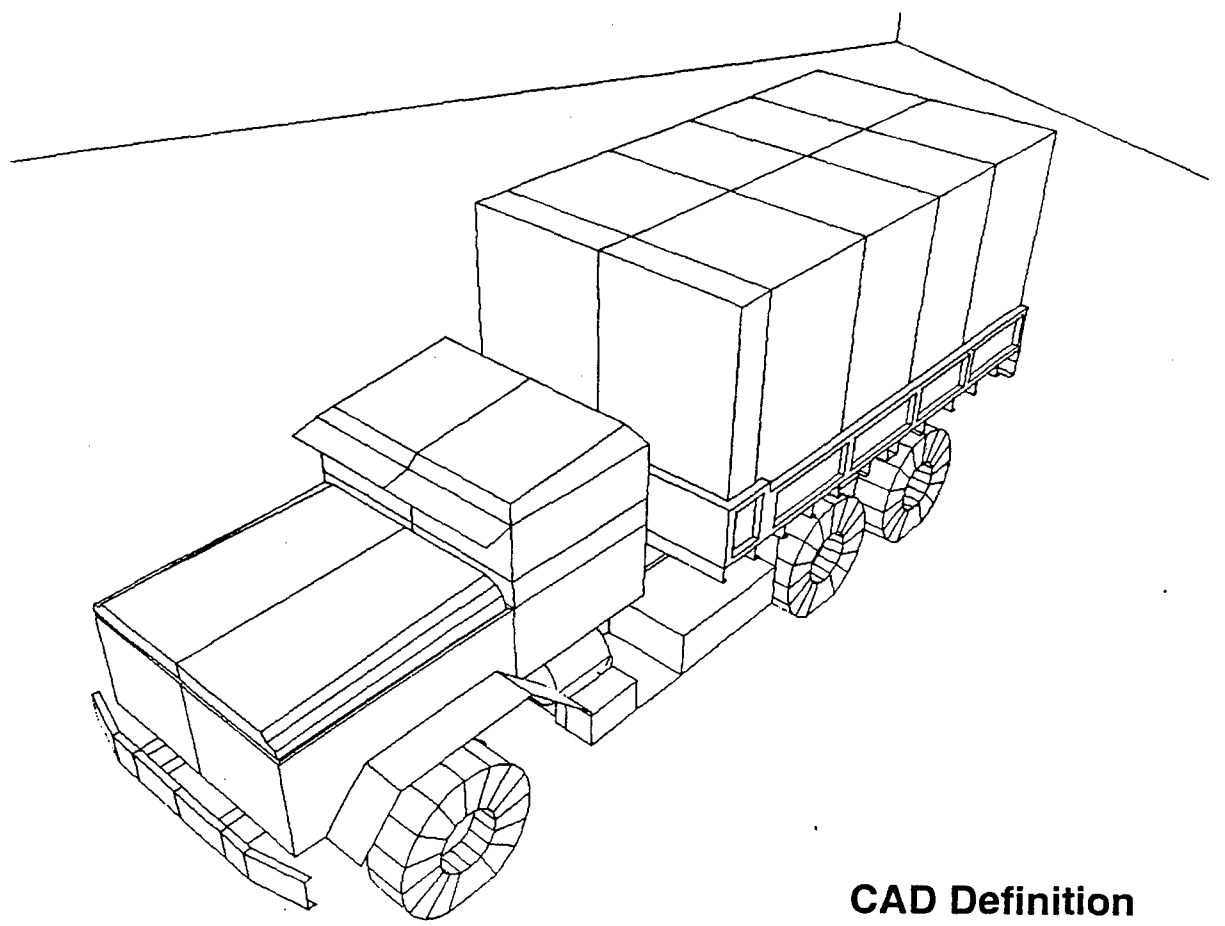
f) Comparison to Experimental Data: Pressure Time Histories, Room 1, Walls; Experimental Data: Thin Lines, Numerical Predictions: Thick Lines



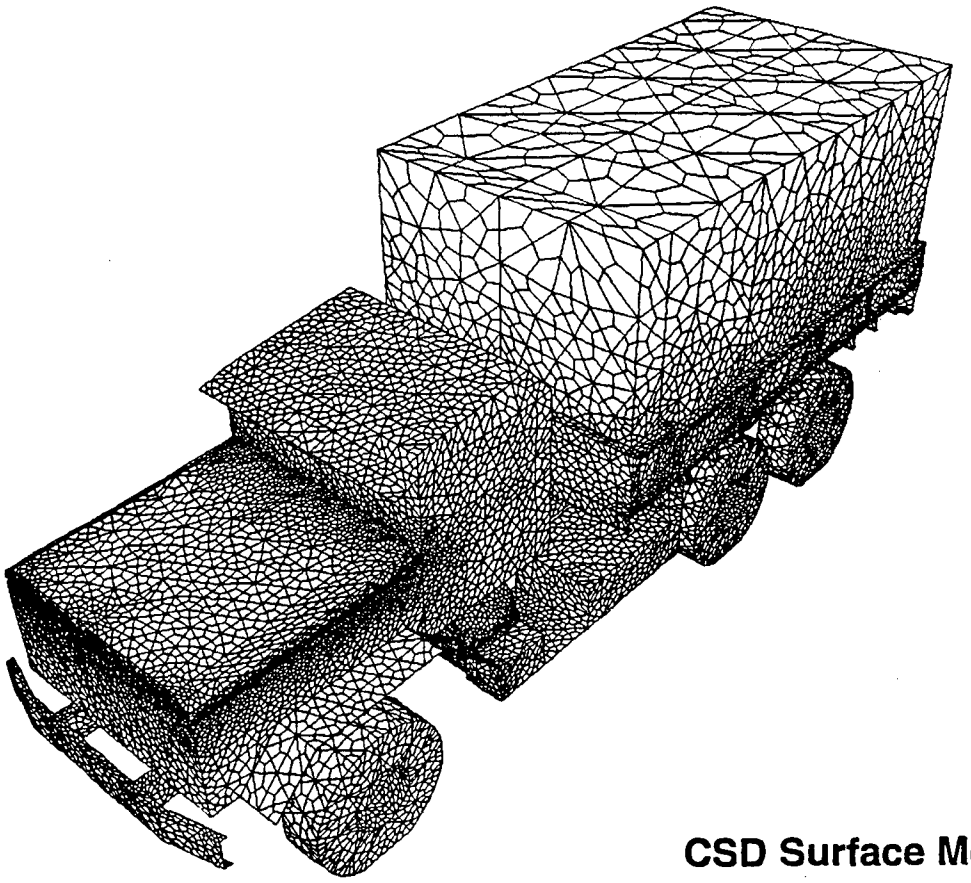
g) Comparison to Experimental Data: Displacement Time History, Wall Between Rooms 1,2.

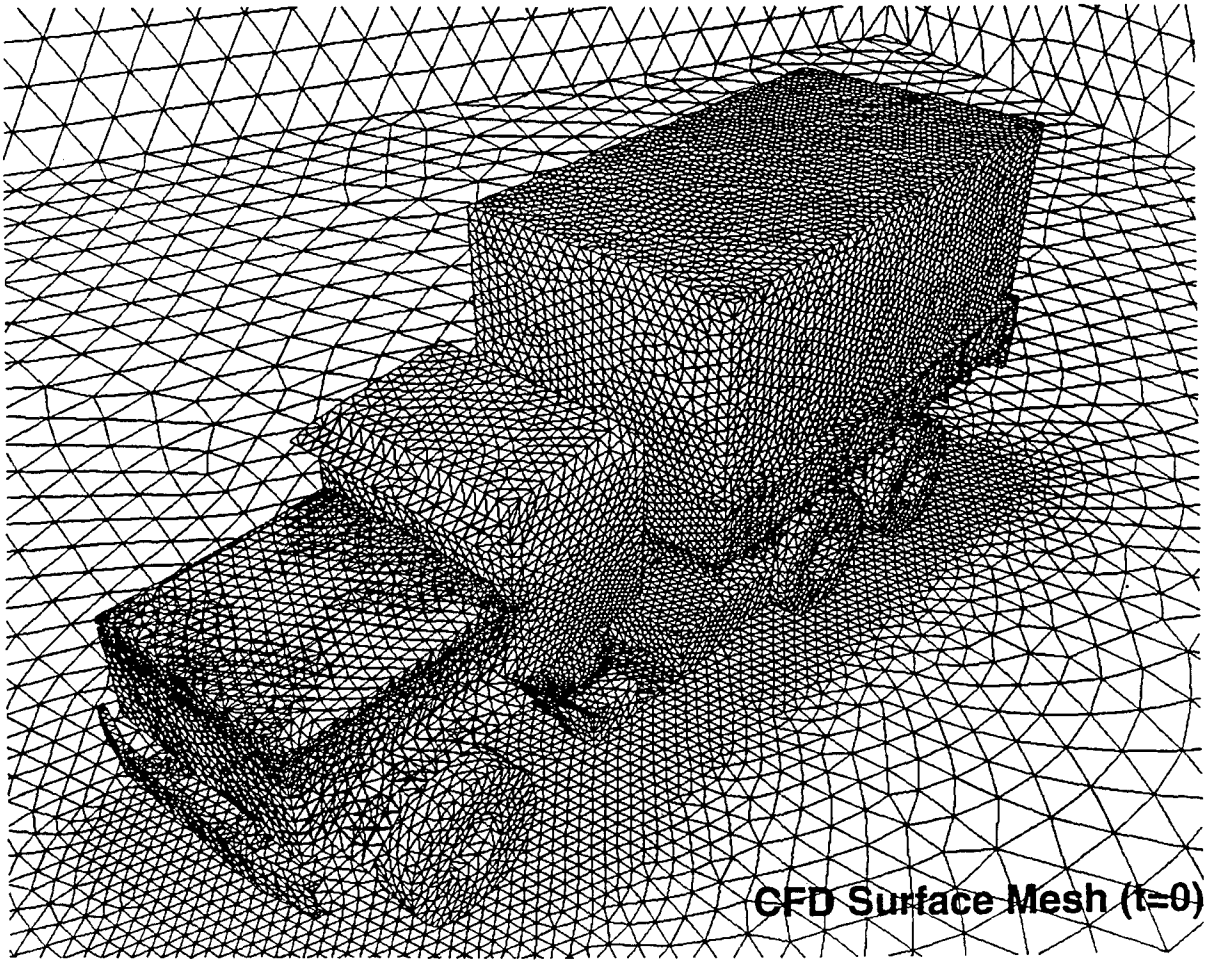
Figure 13: Shock-Truck Interaction





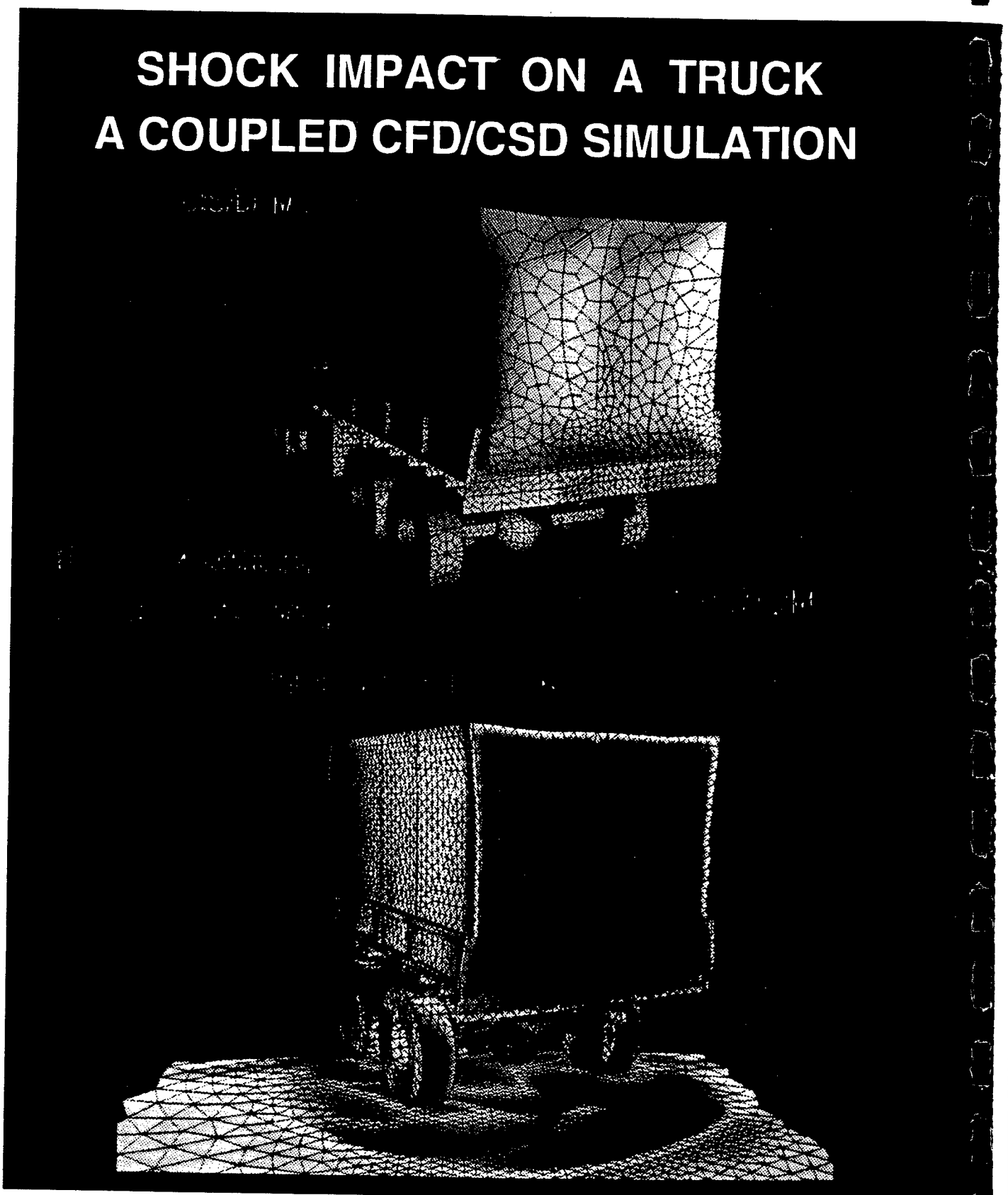
a) Problem Definition



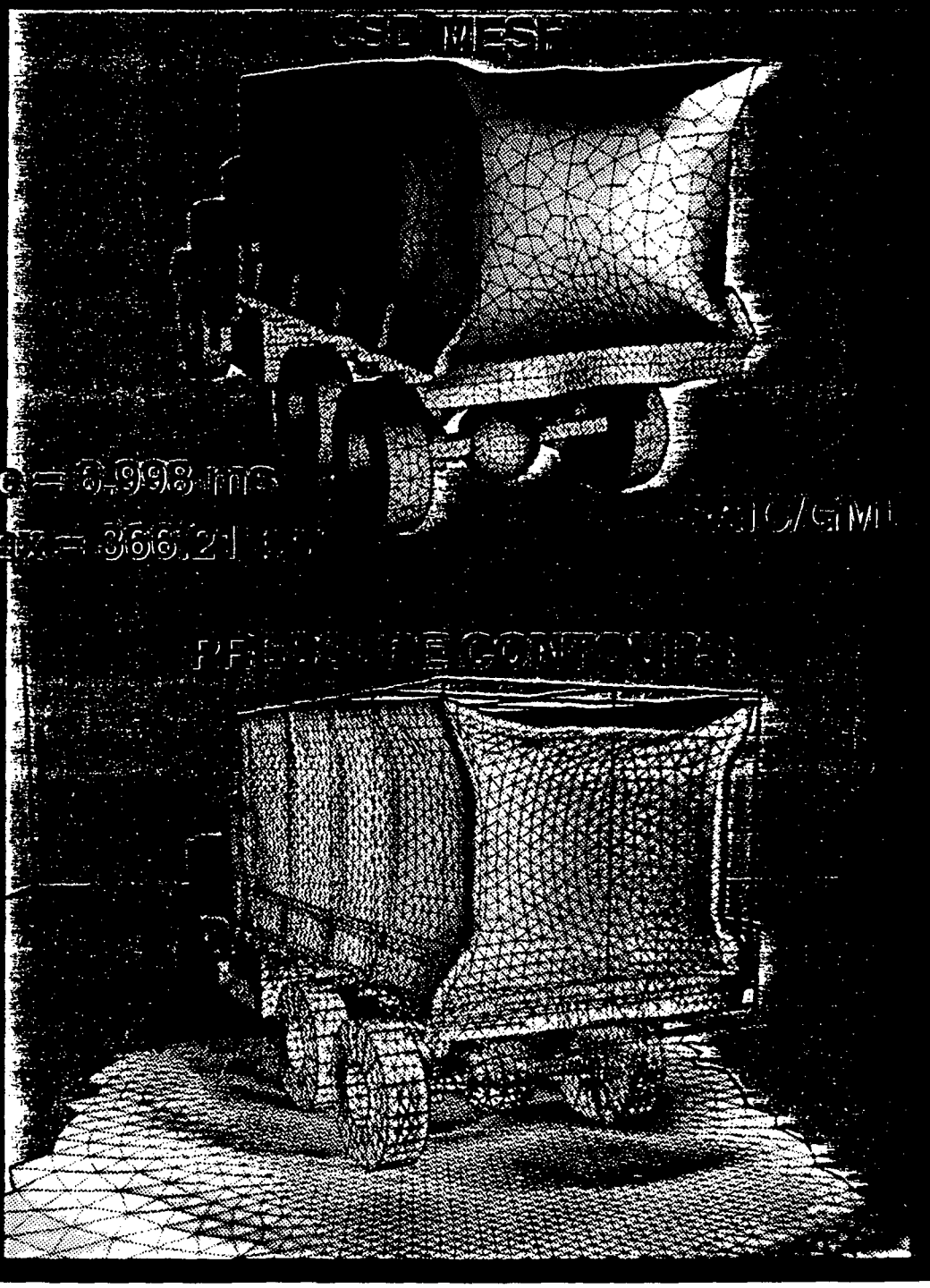


b) Surface Grids

SHOCK IMPACT ON A TRUCK
A COUPLED CFD/CSD SIMULATION



SHOCK IMPACT ON A TRUCK
A COUPLED CFD/CSD SIMULATION



APPENDIX 6: SURFACE MESHING FROM DISCRETE DATA

JOURNAL OF COMPUTATIONAL PHYSICS 126, 1–10 (1996)
ARTICLE NO. 0115

Regridding Surface Triangulations

RAINALD LÖHNER

GMU/CSI, The George Mason University, Fairfax, Virginia 22030-4444

Received August 11, 1995; revised December 26, 1995

An advancing front surface gridding technique that operates on discretely defined surfaces (i.e. triangulations) is presented. Different aspects that are required to make the procedure reliable for complex geometries are discussed. Notable among these are (a) the recovery of surface features and discrete surface patches from the discrete data, (b) filtering based on point and side-normals to remove undesirable data close to cusps and corners, (c) the proper choice of host faces for ridges, and (d) fast interpolation procedures suitable for complex geometries. Post-generation surface recovery or repositioning techniques are discussed. Several examples ranging from academic to industrial demonstrate the utility of the proposed procedure for *ab initio* surface meshing from discrete data, such as those encountered when the surface description is already given as discrete, the improvement of existing surface triangulations, as well as remeshing applications during runs exhibiting significant change of domain.

1. INTRODUCTION

The first and by far the most tedious step of any mesh generation procedure is the definition of the boundaries of the domain to be gridded. This may be accomplished in two ways: (a) analytically, i.e. via functions, or (b) using a tesselation or triangulation. From a practical point of view, it would seem that an analytic definition of the surface is the method of choice, given that nowadays most engineering data originates from some CAD-CAM package. However, in many instances, the boundaries of the domains to be gridded are not defined in terms of analytical functions, such as splines, B-splines, Coon's patches, or NURBS surfaces, but in terms of a triangulation, i.e., *discrete faces and points*. Several classes of applications where this is the case include:

—Visualization and manipulation of complex analytical functions, such as implicit analytic surfaces obtained via superposition or convolution [1, 2];

—Numerical simulations with geometric input data from measurements, such as

—Climate modeling, where the surface of the earth is available from remote sensing data;

—Groundwater and seepage modelling, where the geological layers have been obtained from drill data or seismic analysis; and

—Medical problems, where the patient data has been obtained from CAT scans;

—Numerical simulations that require remeshing, either

—Within the same field solver (e.g., forging simulations, where remeshing is required to regularize the grid, or simulations with adaptive remeshing); or

—For use with a different field solver (e.g., fluid–structure interaction problems, where the surface of the fluid domain is given by the structural surface grid [3], or hypersonic reentry problems, where ray-tracing based on the CFD mesh is used for heat transfer calculations).

Given this discrete data, one may either approximate it via analytical functions, or work directly with it. We prefer the second choice, as the proper approximation via analytical functions becomes cumbersome and problematic for complex geometries. A further reason for using directly discrete data is the fact that surface intersection and trimming are much easier on discrete data than on analytic surfaces. This allows the concurrent generation of surfaces by different users, that are then merged quickly to obtain the final configuration [4].

The present paper describes a surface meshing procedure for discrete data that employs the advancing front technique [5–12]. The technique is based on three steps: surface feature recovery, actual gridding, and surface recovery. The outline of the paper is as follows: having given the rationale for surface meshing *ab initio* from discrete data, Section 2 treats the problem of surface feature recovery. This step allows the surface gridding to obey sides, cusps, or other "ridge" features that may be present in the discrete data, and results in discrete surface patches. Section 3 describes the surface gridding of these discrete surface patches via the advancing front technique. Section 4 considers ways of making the procedure robust in the presence of sharp corners or convoluted patches. Section 5 considers the surface to surface interpolation problem, which is of fundamental importance for large surface grids, and Section 6 the postgeneration surface recovery. In Section 7 several examples that demonstrate the versatility and utility of the procedure are given. Finally, some conclusions are drawn and an outlook for future work is presented in Section 8.

0021-9991/96 $18.00

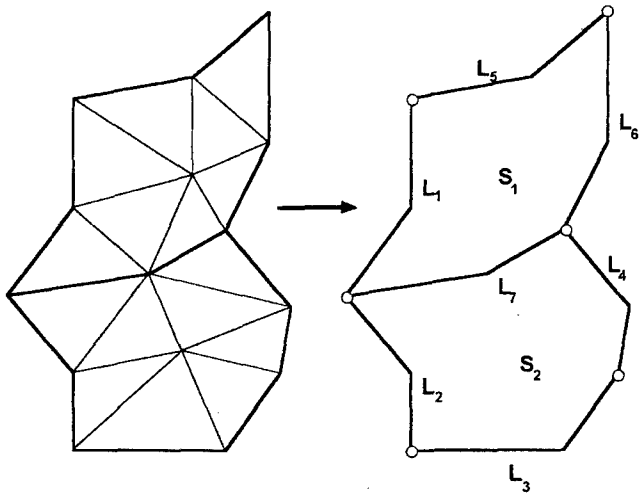


FIG. 1. Discrete surface patch recovery.

2. SURFACE FEATURE RECOVERY

A basic requirement for any surface gridder is that it obey sides, cusps, or other "ridge" features that may be present in the actual surface. In order to avoid gridding over these "ridge" features, sides are first generated along them, and then the surface is gridded with this initial front of sides. A simple way to determine ridges is by comparing the unit surface normals of adjacent faces. If the scalar product of them lies below a certain tolerance, a ridge is defined. *Corners* are defined as points that are attached to:

—Only one ridge;

—More than two ridges; or

—Two ridges with considerable deviation of unit side-vector.

Between corners, the ridges form *discrete lines*. These discrete lines either separate or are embedded completely (i.e., used twice) in *discrete surface patches*. The formation of discrete surface patches is performed with an advancing front algorithm. An arbitrary surface face is selected as a starting face and assigned a patch number. All neighbours that are not separated by a ridge are kept in a local list. The faces of this local list are interrogated for free neighbours in turn and are assigned the current patch number. This local list of neighbour faces becomes empty once all the contiguous faces not separated by a ridge have been marked. This procedure is repeated for all unmarked faces, yielding a list of patches. Using the information of which sides belong to a face, the discrete lines can be assigned to the patches in turn. Figure 1 sketches the recovery of surface features and the definition of discrete surface patches for a simple configuration.

3. ADVANCING FRONT TECHNIQUE

The advancing front technique has gained widespread acceptance for grid generation due to its versatility and speed [5–12]. The basic technique consists in marching into the as yet ungridded region by adding one face at a time. The border separating the gridded region from the as yet ungridded one is called the front. The algorithm may be summarized as follows:

F1. Define the surfaces to be gridded. In the present case, this is done via triangulations. At the same time, define the boundaries of the surfaces.

F2. Define the spatial variation of element size, stretchings, and stretching directions for the elements to be created.

F3. Using the information given for the distribution of element size and shape in space, as well as the line-definitions: generate sides along the lines that connect surface patches. These sides form an initial front for the triangulation of the surface patches.

F4. Select the next side to be deleted from the front; in order to avoid large faces crossing over regions of small faces, the side forming the smallest new face is selected as the next side to be deleted from the front.

F5. Determine the discrete surface face IFADS that contains or is close to the midpoint of the side to be deleted.

F6. Obtain the unit surface normal $\mathbf{n}_{fds}$ for IFADS.

F7. With the information of the desired element size and shape, and $\mathbf{n}_{fds}$: Select a "best point" position for the introduction of a new point IPNEW (see Fig. 2).

F8. Determine whether a point exists in the already generated grid that should be used in lieu of the new point. If there is such a point, set this point to IPNEW and continue searching.

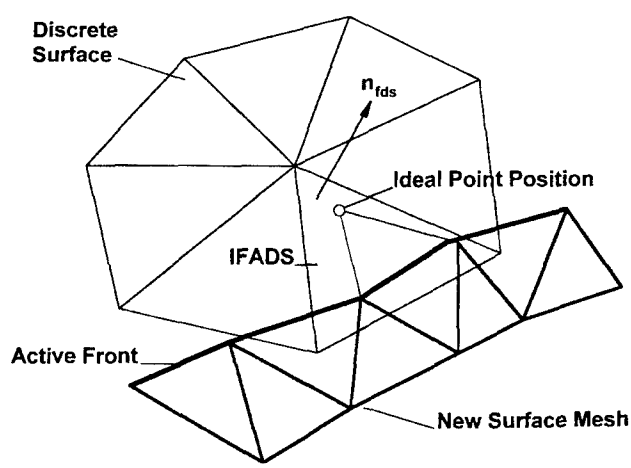


FIG. 2. Generation of surface triangulation on discrete surface.

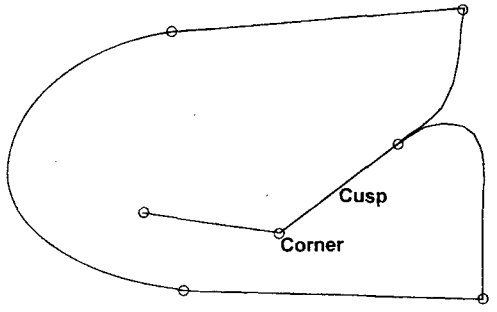


FIG. 3. Discrete surface patch with cusps and corners.

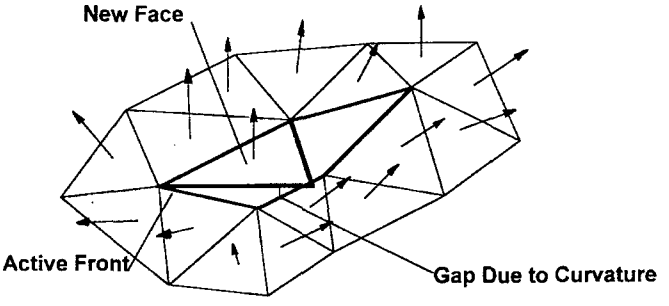


FIG. 4. Non-uniqueness of front crossing for curved surfaces.

F9. Determine whether the face formed with the selected point `IPNEW` does not cross any given sides. If it does. select a new point as `IPNEW` and try again.

F10. Add the new face, point, and sides to their respective lists.

F11. If a new point was added: reposition it on the discretely defined surface.

F12. Find the desired element size and stretching for the new sides.

F13. Delete the known sides from the list of sides.

F14. If there are any sides left in the front, go to F4.

As compared to the surface gridding of analytically defined surfaces, which has been treated by a number of authors [7–9, 11, 12], the only differences are:

—The search for the discrete surface face containing or close to a point (Steps F5, F11);

—The introduction of a normal vector $\mathbf{n}_{fds}$ for each face in order to determine the ideal point position (Step F7);

—The repositioning of new points on the discretely defined surface (F11).

4. ENHANCEMENTS FOR ROBUSTNESS

The procedure, as described above, will work well for smooth surfaces. In practice, however, one is often faced with discrete surfaces that exhibit cusps, sharp corners, or ridges with high curvature (see Fig. 3). In these instances, the procedure must be enhanced in order to work reliably. The most important of these enhancements: 2D crossing check, point and side normals, angle of visibility for filtering inappropriate data and proper host face for ridges, are described in the sequel.

4.1. *2D crossing check.* In regions of colliding fronts on 3D surfaces with curvature, the face/side-crossing check is not uniquely defined (see Fig. 4). This ambiguity is best circumvented by transforming all the points required, i.e., those in the list of close points and those attached to close sides, to the plane defined by the midpoint of the side to be deleted and the normal vector $\mathbf{n}_{fds}$. Thereafter, the face/side-crossing check is performed in 2D.

4.2. *Point and side normals.* It is advisable to use the point and side normals obtained by interrogating the host face of the discrete data in order to filter out undesired data from the list of close points and sides. In this way, the front data of the lower portion of the cusp shown in Figure 3 is automatically removed when generating a face on the upper portion and vice versa.

4.3. *Angle of visibility.* In order to avoid the improper choice of close wrong points that may have the correct point and side normals, but belong to another portion of the discrete surface patch (see Fig. 5), all points outside the allowable "angle of visibility" α are no longer considered. A meaningful value for α can be obtained by measuring the local surface curvature of the underlying discrete surface in the vicinity of the side to be removed from the front. In the present case, this is done by simply comparing the normals of the host face and its neighbours.

4.4. *Proper host face for ridges.* For the sides along ridges, there can be instances where the host face is not properly defined. As an example, consider the situation

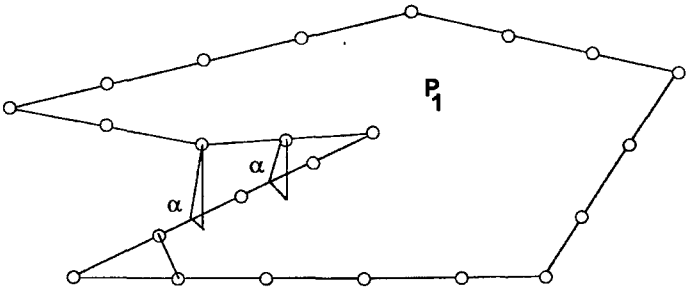


FIG. 5. Filtering with angle of visibility.

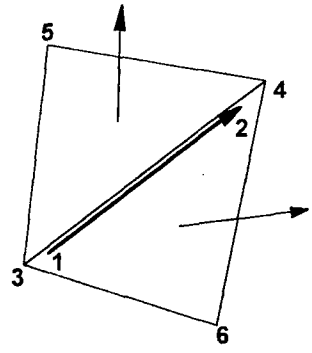


FIG. 6. Selection of proper host faces at ridges.

shown in Fig. 6. Given the orientation of the side 1-2, the proper host face is 3-4-5. However, face 3-6-4 could also be considered as a host face. In order to resolve this ambiguity, the point $\mathbf{x}_{\text{fds}}$ that is furthest from the side is determined for each possible host face. The proper host face has to satisfy

$$c_s = [(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_{\text{fds}} - \mathbf{x}_1)] \cdot \mathbf{n}_{\text{fds}} > 0. \tag{1}$$

5. SURFACE TO SURFACE INTERPOLATION

One of the main differences between gridding discrete, as opposed to analytic, surfaces is the potentially very expensive search for the host face of each new point and side generated. Careless implementation of these operations would lead to an $O(N_p \cdot N_{dp})$ complexity, where N_p denotes the number of new surface points created and N_{dp} the number of points defining the discrete surface patch. If both of these are of similar magnitude, the result is a complexity of $O(N_p^2)$, clearly inappropriate for large surface grids. Given that the advancing front algorithm by its very nature adds points and sides in the vicinity of known data points, the host face of the side to be taken out can be used as a good starting guess from which to find the correct host face via a neighbour-to-neighbour search (see

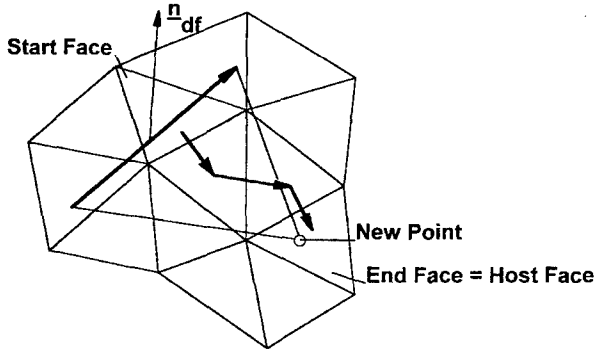


FIG. 7. Neighbour-to-neighbour jump search procedure.

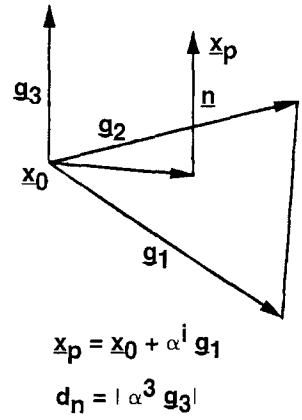


FIG. 8. Surface interpolation.

Fig. 7). With the notation of Fig. 8, the point to be interpolated $\mathbf{x}_p$ is given by

$$\mathbf{x}_p = \mathbf{x}_0 + \sum_{i=1}^{3} \alpha^i \mathbf{g}_i, \tag{2}$$

where

$$\mathbf{g}_i = \mathbf{x}_i - \mathbf{x}_0, \quad i = 1, 2; \quad \mathbf{g}_3 = \frac{\mathbf{g}_1 \times \mathbf{g}_2}{|\mathbf{g}_1 \times \mathbf{g}_2|}, \tag{3a), (3b)}$$

and the shape-functions or barycentric coordinates N^i are given by

$$N^i = \alpha^i, \quad i = 1, 2; \quad N^0 = \alpha^0 = 1 - \alpha^1 - \alpha^2, \tag{3c), (3d)}$$

the point $\mathbf{x}_p$ is considered as being on the surface face **Iff**:

$$\min(N^i, 1 - N^i) \geq 0, \quad \forall i = 0, 1, 2, \tag{4a}$$

and

$$d_n = |\alpha^3 \mathbf{g}_3| \leq \delta_n. \tag{4b}$$

Here δ_n denotes a tolerance for the relative distance normal to the surface face. Many search and interpolation algorithms have been devised over the years. We have found that for generality, a layered approach of different interpolation techniques works best. Wherever possible, a vectorized advancing front neighbour-to-neighbour algorithm is employed as the basic procedure [13]. Given that the advancing front algorithm by its very nature adds points and sides in the vicinity of known data points, the host face of the side to be taken out can be used as a good starting guess from which to find the correct host face via this neighbour-to-neighbour search (see Fig. 7). Should this fail, octrees [14, 15] are employed. Finally, if this ap-

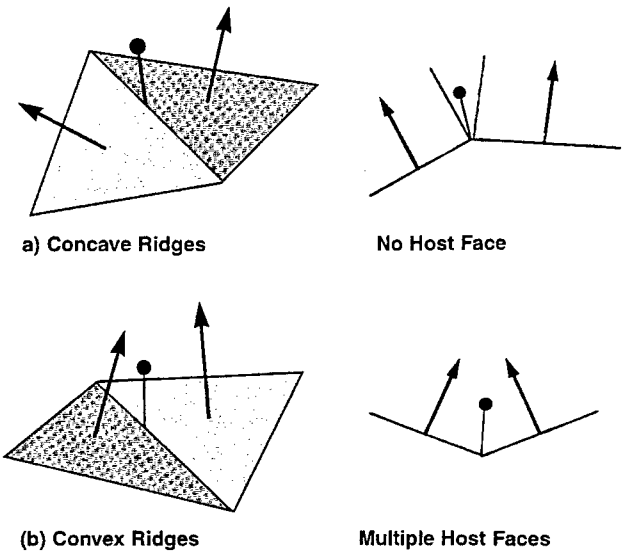


FIG. 9. Problems when searching for host faces.

proach fails too, a brute force search over all the surface faces is performed [13]. For realistic 3D surface geometries, the interpolation of surface grid information may be complicated by a number of the factors. The first of these factors is the proper choice of δ_n, i.e., the proper answer to the question: "How close must a face be to a point in order to be acceptable ?" This is not a trivial question for situations where narrow gaps exist in the discrete surface mesh, when there is a large discrepancy of face-sizes between the discrete surface grids and the new surface grid, as well as when the discrete surface grid exhibits highly stretched elements. Our experience indicates that the choice

$$\delta_n < c_n \cdot |\mathbf{g}_1 \times \mathbf{g}_2|^{0.5}, \quad c_n = 0.05, \tag{5}$$

works reliably, although the constant c_n may be problem dependent. A second complication often encountered arises due to the fact that Eq. (4a) may never be satisfied (e.g., the convex ridge shown in Fig. 9a), or it may be satisfied by more than one surface face (e.g., the concave ridge shown in Fig. 9b). In the first instance the criterion given by Eq. (4a) may be relaxed somewhat to

$$\min(N^i, 1 - N^i) \geq \varepsilon, \quad \forall i = 0, 1, 2, \tag{6}$$

where ε is a small number. For the second case, the discrete surface face with the smallest normal distance d_n is selected. We remark that in both of these instances the final point location is unaffected by the final host surface face, as the interpolation weights are such that only the points belonging to the ridge are used for interpolation. We have found that it is very important to take the face that has the smallest distance to the point being interpolated in order to mitigate any possible problems. For situations close to corners, gaps, or multi-surface configurations, an exhaustive search over all faces will be triggered. In order not to check in depth the complete surface mesh, only the faces that satisfy the relaxed closeness criteria $\varepsilon \geq -1$, $c_n \leq 0.5$ are considered. The face with the closest distance to the point is kept. If a face satisfies Eq. (4a), the closest distance is indeed d_n. Should this not be the case, the closest distance to the three sides ij of the face is taken:

$$\delta = \min_{ij}|\mathbf{x}_p - (1 - \beta_{ij})\mathbf{x}_i - \beta_{ij}\mathbf{x}_j|,$$
$$\beta_{ij} = \frac{(\mathbf{x}_p - \mathbf{x}_i) \cdot (\mathbf{x}_j - \mathbf{x}_i)}{(\mathbf{x}_j - \mathbf{x}_i) \cdot (\mathbf{x}_j - \mathbf{x}_i)}. \tag{7a), (7b}$$

Should two faces have the same normal distance, the one with the largest minimum shape-function α^i, $i = 0, 1, 2$ is retained.

A third complication arises for cases where cusps or close surfaces are present. For these cases, the "best" face may actually lie on the opposite side of the face being interpolated. This ambiguity is avoided by defining a surface normal, and then only considering the faces and points whose normals are aligned, i.e., those for which

$$\mathbf{n}_{\text{fds}} \cdot \mathbf{n}_p > c_s, \quad c_s = 0.5. \tag{8}$$

Here $\mathbf{n}_{\text{fds}}, \mathbf{n}_p$ denote the discrete surface face and the point/side-normals respectively. Experience indicates that it is advisable to perform a local exhaustive search for all faces surrounding the best host face found in order to obtain the host face that satisfies Eqs. (4), (7) as best possible.

Although these extra steps for interpolation seem complex, they are not only indispensable for discrete data that exhibits cusps, high surface curvature, and internal ridges, but their cost is not significant.

6. POSTGENERATION SURFACE RECOVERY

After the surface grid has been generated, it may be desirable to reposition the points in order to meet certain surface fidelity criteria. Obvious choices, shown in Fig. 10, are:

(a) *Keep as is.* I.e., no postprocessing. This will be the preferred choice if the loss of surface fidelity due to curvature and/or different face-sizes is small.

(b) *Move to closest discrete point.* The rationale for this option is that if the underlying discrete data is of much finer resolution than the newly generated mesh, moving each point to a given point will not distort the mesh significantly while assuring an exact pointwise representation.

(c) *Higher order recovery.* In this case, the new sur-

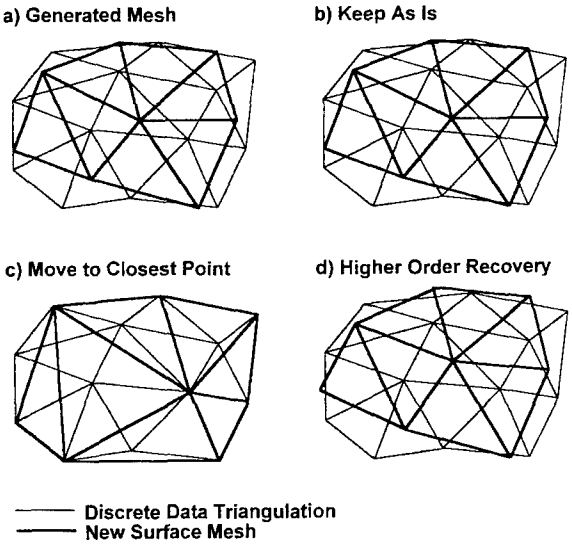


FIG. 10. Postgeneration surface recovery.

face points are repositioned using higher order recovery procedures for the discrete data. This option is attractive if the discrete surface data is much coarser than the newly generated mesh and exhibits surface curvature. The first step consists in computing average normals at the points of the discrete surface. In a second step, the information of which host face a point belongs to, and which are its local area coordinates ζ_1, ζ_2, ζ_3, together with the point normals, is used to reposition the point. We have considered to date:

(c1) *Quadratic recovery.* For each side of the discrete surface triangulation, a mid-point location is estimated from a Hermitian polynomial as

$$\mathbf{x} = (1-\xi)^2(1+2\xi)\mathbf{x}_1 + \xi(1-\xi)^2\mathbf{r}_1 + \xi^2(3-2\xi)\mathbf{x}_2 - \xi^2(1-\xi)\mathbf{r}_2, \tag{9}$$

where

$$\mathbf{r} = s\frac{\mathbf{n}\times(\mathbf{s}\times\mathbf{n})}{|\mathbf{n}\times(\mathbf{s}\times\mathbf{n})|}, \quad \mathbf{s} = \mathbf{x}_2 - \mathbf{x}_1, \quad s = |\mathbf{s}|, \tag{10}$$

and $\xi = 0.5$. With this information, and using the notation in Fig. 11, the recovered point location is given by the standard quadratic triangle shape functions [16]:

$$\mathbf{x} = \zeta_1(2\zeta_1-1)\mathbf{x}_1 + \zeta_2(2\zeta_2-1)\mathbf{x}_2 + \zeta_3(2\zeta_3-1)\mathbf{x}_3 + 4\zeta_1\zeta_2\mathbf{x}_4 + 4\zeta_2\zeta_3\mathbf{x}_5 + 4\zeta_3\zeta_1\mathbf{x}_6. \tag{11}$$

(c2) *Cubic recovery.* For each face of the discrete surface triangulation, we have nine pieces of information: location of the end-points, and inclination of the normals with respect to the plane formed by the plane. For a complete cubic, 10 pieces of information are required. We use the Zienkiewicz triangle, derived for plate elements [16], to account for this deficit.

The recovery procedures described represent just two instances of many possible alternatives, such as mid-normals, local spline, Clugh–Tocher, Doo–Sabin, etc. [17, 18].

7. EXAMPLES

The described procedure was applied to a number of cases in order to test its applicability in production environments. For all of these cases, the surface was recovered using quadratic and cubic functions. However, no graphically discernable difference was encountered.

6.1. *Sphere.* We start with this academic example to show the basic possibilities of surface gridding based on discrete data representations. An initial surface mesh, shown in Fig. 12a, is taken as the starting point. This mesh contains no faces whose normals vary by more that $\beta = 10°$ among neighbours. For this reason, the whole surface is treated as one patch, with the largest side taken as a discrete line. Two new surface grids, one of constant element size and one with a prescribed source at one end of the sphere, were generated and are shown in Fig. 12b, c.

6.2. *Forging piece.* This second problem demonstrates the use of surface remeshing for discretely defined domains within the same numerical simulation. A piece that originally started out as pie-shaped has been distorted significantly due to forging. A complete remeshing of the computational domain is required. The surface of the mesh at this stage is shown in Fig. 13a. The edge-detection algo-

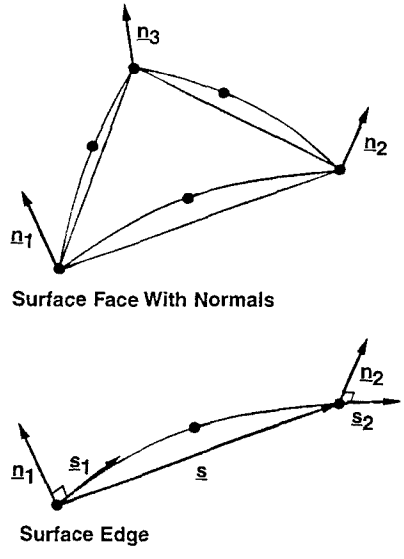


FIG. 11. Quadratic surface reconstruction.

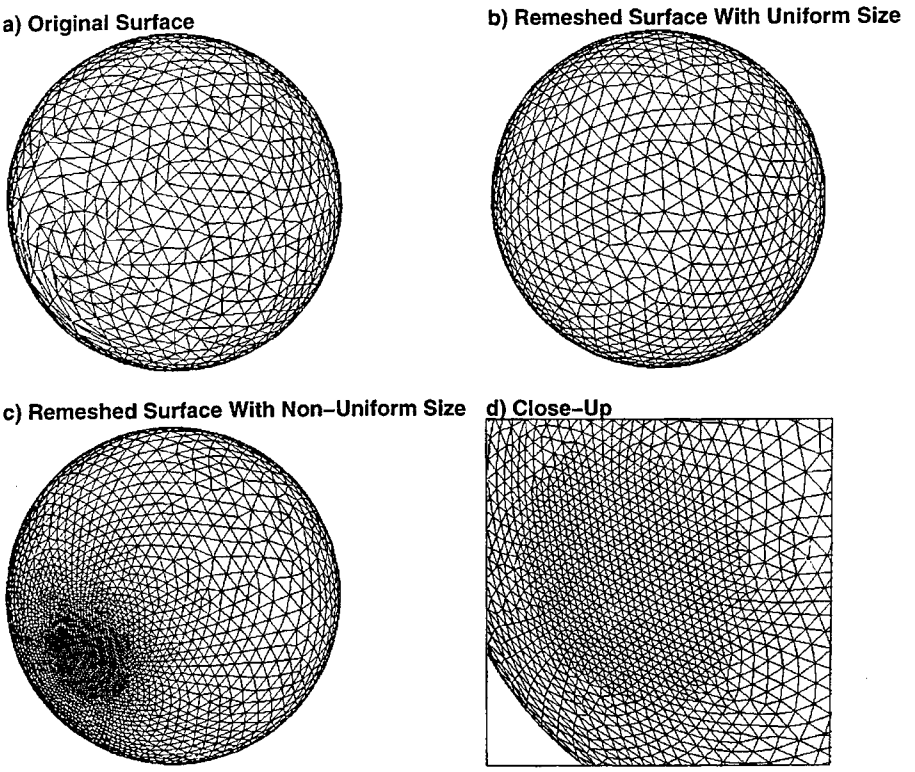


FIG. 12. Sphere.

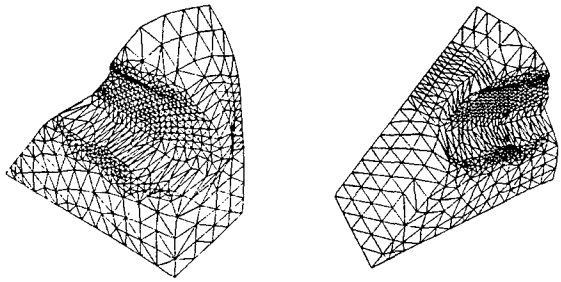
rithm then forms the discrete line and surface patch definition shown in Fig. 13b. The angle used to determine ridges was set to $\beta = 25°$. An adaptive background grid is generated automatically [10], starting from a cube. With this background mesh, a new surface triangulation, shown in Fig. 13c, is generated.

6.3. *Ship.* This case illustrates the use of surface meshing from discrete data as a means to expedite the domain definition process, as well as the possibility of correcting an initially improper surface discretization. An initial surface mesh for the ship, shown in Fig. 14a, was provided as a starting point. The discrete lines and surface patches obtained using an angle tolerance of $\beta = 30°$ between adjacent faces are shown in Fig. 14b. A finer meshing region close to the water line was specified by using two surface sources [10]. The final surface mesh, given in Fig. 14c, not only exhibits a better discretization (i.e., less small angles), but it is also better suited for the numerical simulation.

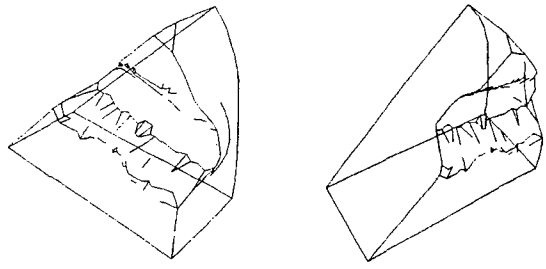
6.4. *Car fender die.* This case shows the use of surface meshing from discrete data as a means of streamlining data input within industrial simulations. The original CAD dataset had over 500 surface patches, many of them overlapping and in need of trimming. Instead, a cloud of points, obtained from a digitization of the actual part, is taken as the starting point. This cloud of approximately 5,000 points, shown in Fig. 15a, is then triangulated using an automatic surface recovery tool developed by the author [19] (for automatic surface recovery, see also [18, 20]). The surface is now defined discretely, and lines and patches, shown in Fig. 15c, are recovered. The final surface mesh, suitable for stamping calculations, is shown in Fig. 15d. This example clearly demonstates the possible advantages of discrete surface gridding. Trimming and combining over 500 surfaces is a tedious and time-consuming effort, which can be reduced drastically as shown here.

6.5. *Generic hypersonic airplane geometry.* This final case shows the combination of discrete and analytically defined surfaces to obtain rapid turnaround in preliminary design calculations. The airplane fuselage is given from a structural dynamics calculation and shown in Fig. 16a. The recovered discrete surface patches, together with the added outer box and some further analytical patches for nozzle entry and exit planes, is shown in Fig. 16b. The new surface discretization, suitable for preliminary aerodynamic design calculations, is shown in Fig. 16c.

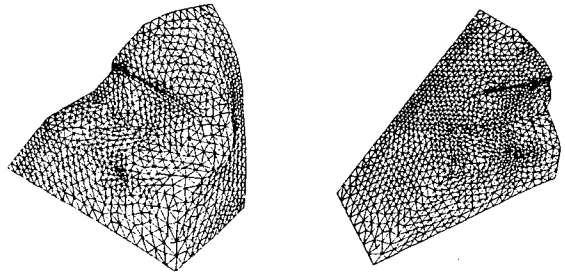
All of the surface grids shown were obtained in less than 5 min on an IBM RISC-550 workstation, indicating that it is feasible to port these automatic surface meshing and remeshing techniques into production codes.



Original Surface Mesh



Discrete Lines and Surface Patches



Surface Mesh With Background Grid Adaptation

FIG. 13. Forging piece.

8. CONCLUSIONS AND OUTLOOK

An advancing front surface gridding technique that operates on discretely defined surfaces has been presented. Different aspects that are required to make the procedure reliable for complex geometries are discussed. These include:

—Recovery of surface features and discrete surface patches;

—Filtering based on point and side-normals to remove undesirable data close to cusps and corners;

—Filtering based on an angle of visibility to remove irrelevant close-point/side data;

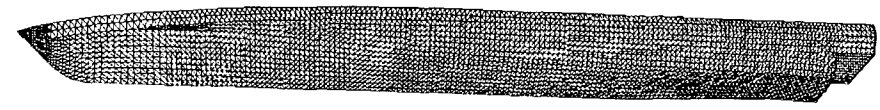
—The proper choice of host faces for ridges; and

—Fast interpolation procedures suitable for complex geometries.

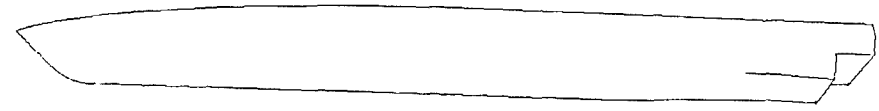
The task of postgeneration surface recovery or repositioning is also discussed, and some of the many possible alternatives are given.

Several examples ranging from academic to industrial demonstrate the utility of the developed procedure for *ab initio* surface meshing from discrete data, such as those encountered when the surface description is already given as discrete, the improvement of existing surface triangulations, as well as remeshing applications during runs exhibiting significant change of domain.

As with any other technique, improvements are always possible. They will center on better postgeneration surface recovery schemes and further enhancements in robustness and reliability for complex geometries.



a) Original Surface Grid



b) Discrete Lines and Surface Patches

c) New Surface Grid

FIG. 14. Ship hull.

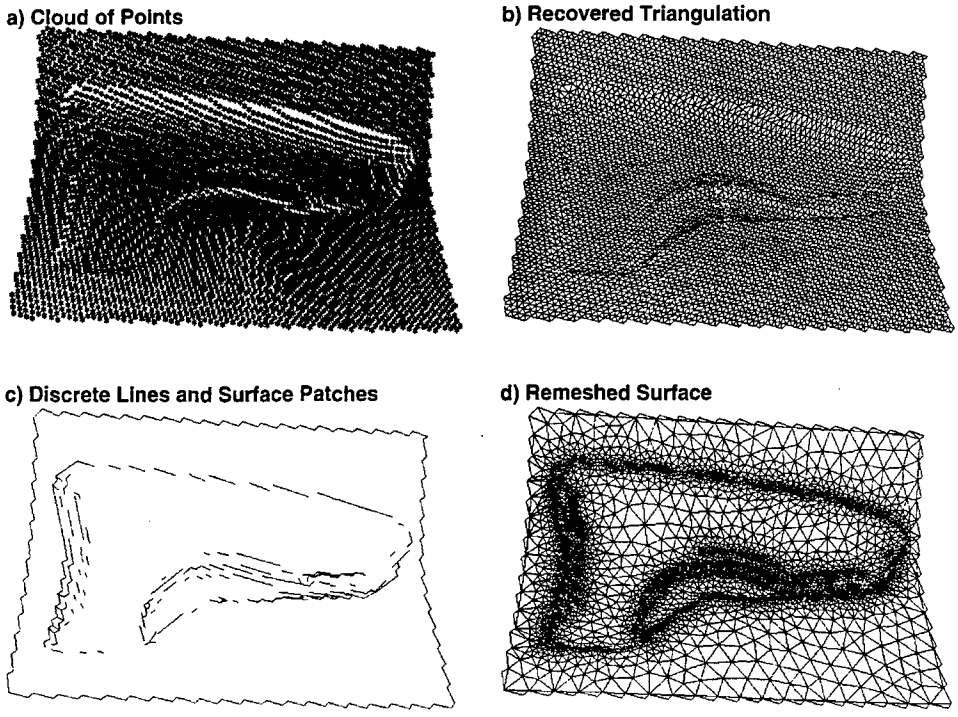


FIG. 15. Car fender die.

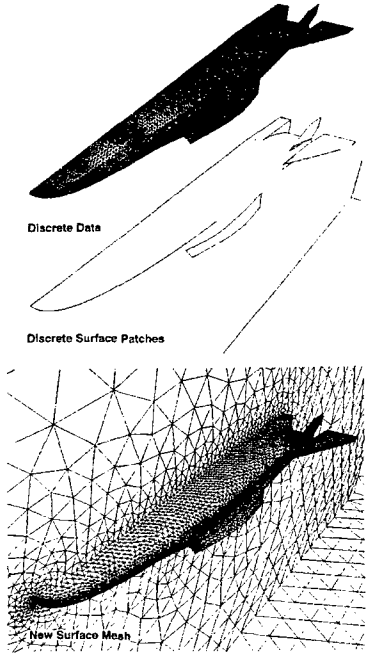


FIG. 16. Generic hypersonic airplane geometry.

ACKNOWLEDGMENTS

A considerable portion of this work was carried out while the author was visiting the Centro Internacional de Métodos Numericos en Ingenieria (CIMNE) at the Universidad Politécnica de Catalunya, Barcelona, Spain. The support for this visit is gratefully acknowledged.

REFERENCES

1. J. Bloomenthal, *Comput. Aided Geom. Design*, November (1988).
2. J. Bloomenthal and K. Ferguson, *Proc. SIGGRAPH*, August (1995).
3. R. Löhner, C. Yang, J. Cebral, J. D. Baum, H. Luo, D. Pelessone, and C. Charman, AIAA-95-2259, (1995) (unpublished).
4. S. H. Lo, *Int. J. Numer. Methods Eng.* **38,** 943 (1995).
5. J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz. *J. Comput. Phys.* **72,** 449 (1987).
6. R. Löhner, *Commun. Appl. Numer. Methods* **4,** 123 (1988).
7. R. Löhner and P. Parikh, *Int. J. Numer. Methods Fluids* **8,** 1135 (1988).
8. J. Peiro, J. Peraire, and K. Morgan, *Proceeding. POLYMODEL XII Conf., Newcastle-upon-Tyne, May 23–24* (1989) (unpublished).
9. J. Peraire, K. Morgan, and J. Peiro, AGARD-CP-464. 18, (1990) (unpublished).
10. R. Löhner, *Commun. Appl. Numer. Methods*, submitted.
11. K. Nakahashi and D. Sharov, AIAA-95-1686-CP, 1995 (unpublished).
12. C.-J. Woan, AIAA-95-2202, 1995 (unpublished).
13. R. Löhner, *J. Comput. Phys.* **118,** 380 (1995).
14. D. N. Knuth, *The Art of Computer Programming*, Vol. 3 (Addison–

Wesley, Reading, MA, 1973).
15. R. Sedgewick, *Algorithms* (Addison–Wesley, Reading, MA, 1983).
16. O. C. Zienkiewicz, *The Finite Element Method*, (McGraw–Hill, New York, 1982).
17. G. Farin, *Comput. Aided Geom. Design* **3**(2), 83 (1986).
18. J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design* (Peters, 1993).
19. R. Löhner, Surface reconstruction from clouds of points, preprint.
20. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Comput. Graph.* **26**(2), 71 (1992).

APPENDIX 7: PROGRESS IN GRID GENERATION

Engineering with Computers (1996) 12: 186–210

Engineering with Computers

Progress in Grid Generation via the Advancing Front Technique

R. Löhner
The George Mason University, VA, USA

Abstract. *We describe recent extensions and improvements to the advancing front grid generation technique. These improvements target a range of applicability, speed and user friendliness. The range of applicability is enlarged by the ability to produce volumetric grids around thin surfaces (such as shells, membranes, fabrics or surfaces with cusps), the generation of high aspect ratio grids for Navier–Stokes applications, the generation of higher order triangular and tetrahedral elements, and the generation of quadrilateral and hexahedral elements. Speed improvements are the result of reduced search overheads, as well as vectorization and parallelization. User friendliness is enhanced by the ability to grid directly discrete data and simpler ways of specifying the desired element size and shape in space. Numerous examples are included that demonstrate the versatility and maturity that advancing front grid generators have achieved.*

Keywords. Finite elements; Grid generation; Tetrahedra; Unstructured grids

1. Introduction

The general topic of unstructured grid generation techniques has seen a major burst of activity in recent years. While only a decade ago the automatic generation of grids for complex geometries in excess of a million elements was impossible, we nowadays commonly deal with problems of this size [1–3]. When contemplating the generation of a grid, four data items must be specified:

(a) A description of the bounding surfaces of the domain to be discretized.
(b) A description of how the element size, shape and orientation should be in space.
(c) The choice of element type.
(d) The choice of a suitable method to achieve the generation of the desired mesh.

Correspondence and offprint requests to: R. Löhner, GMU, CSI, The George Mason University, Fairfax VA 22030-4444, USA.

If we consider the fourth item on this list, i.e. the task of filling a given domain with elements, there appear to be only two basic ways of accomplishing it:

—by filling the 'empty', i.e. as yet ungridded region with elements; or
—by modifying an existing grid that already covers the domain to be gridded.

The first class of techniques denotes the advancing front methods [4–10] the front being defined as the boundary between the gridded and ungridded region. The key algorithmic step that must be addressed for advancing front methods is the proper introduction of new *elements* to the ungridded region. For triangular and tetrahedral grids the elements are introduced sequentially one at a time. For quadrilateral and hexahedral elements, this technique is known as paving or plastering [11, 12]. The second class of techniques is known as Voronoi or Delaunay triangulation methods. Here, the key algorithmic step is the proper introduction of new *points* to the given grid. This class of techniques has been used only for the construction of triangular or tetrahedral grids. The name Voronoi or Delaunay that is associated with these techniques stems from the element reconnection technique most often employed [13–19]. We remark that the modified or finite octree [20, 21] techniques represent just one possible realization of a Delaunay triangulation. Given the known distribution of points from the octree, the mesh connectivity is obtained by applying the circumcircle or Delaunay criterion.

The present paper summarizes some recent extensions and improvements of the advancing front technique. In order to define terms and aid in the understanding of some of the subsequent material, an outline of the basic technique is described in Section 2. Section 3 treats the following extensions in the range of applicability: multimaterial problems, volumetric gridding of thin or crossing surfaces, grids suitable for Navier–Stokes applications, higher order triangles and tetrahedra, and grids consisting only of quadrilateral or hexahedral elements. Speed improvements

combustion simulations, and can take advantage of more accurate discretization schemes.

- Reduction of element count. Today, HEXAR produces meshes that are 2–10 times larger than hand-made meshes.
- Improving implementation in analysis. Our objective is to tune the product so that it can be fully integrated into existing CAE environments. Each specific application area such as structures, cooling, casting or external aerodynamics has its own requirements. HEXAR must be improved in such a way that all these requirements can be satisfied with concern for accuracy and with minimal need for human interaction during mesh generation.
- Reduction of the computational time. Today, HEXAR requires a relatively large amount of resources. This is because HEXAR relies heavily on ray tracing and pattern recognition techniques which can be parallel, but computationally demanding. Future work will involve the exploration of faster methods in order to achieve a hundred-fold reduction in mesh generation time.

6. Conclusion

HEXAR is a good quality, approximate, automatic mesh generator that represents a novel approach to volume mesh generation for CAE. It uses highly parallel algorithms to produce unstructured all-hexahedral meshes featuring local grid refinement and coarsing. It is designed to minimize the length of CAE by reducing the time required to generate meshes. Today, HEXAR produces sometimes a large number of elements, may produce meshes of marginal quality for certain applications and is relatively demanding in terms of computer resources. As it is further developed, its fidelity to the input geometry will improve, its element count will be reduced and its performance will increase substantially.

References

1. Stephenson, M.B., Blacker, T.D. (1989) Using conjoint meshing primitives to generate quadrilateral and hexahedral elements in irregular regions, Computers in Engineering (Riley, D.R., Cokonis, T.J., Gabriele, G., Kinzel, G.L., Tamma, K.K., Bennett, D.W., Kinoglu, M.F., Busnaina, A.A. and Rasdorf, W. Editors), American Society of Mechnical Engineers, Book No. G0502B
2. Taghavi, R.; Dupont, A. (1989) Multidimensional flow simulation in an inlet port/combustion chamber assembly featuring a moving valve, Proceedings of the ASME Energy-Sources Technology Conference and Exhibit, Houston, TX
3. Cabello, J.; Lohner, R.; Jacquotte, O.P. (1991) A variational method for the optimization of two- and three-dimensionas unbstructured meshes, Abstract in the first US National Congress on Computational Mechanics, Chicago, IL

due to reduced search overheads, global h-refinement, vectorization and parallelization are the topic of Section 4. Section 5 deals with improvements in user-friendliness, in particular surface generation from discrete data and more convenient ways of specifying element size and shape for complex geometries. Section 6 is devoted to examples. As most of these examples combine several of the improvements described, the decision was made to add the examples section at the end, instead of showing them 'on the go'. Although this makes the main body of the paper rather 'dry', it should facilitate the overall understanding of the material.

2. The Advancing Front Technique

The advancing front technique [4–10, 22] consists in marching into as yet ungridded space by adding one element at a time. The region separating the gridded portion of space from the as yet ungridded one is called the *front*. The algorithm may be summarized as follows:

F1. Define the boundaries of the domain to be gridded. Without going into further detail, we will assume some general form of hierarchical surface definition consisting of patches, the lines that surround or delimit them, and points at the intersections of lines.

F2. Define the spatial variation of element size, stretchings and stretching directions for the elements to be created.

F3. Using the information given for the distribution of element size and shape in space and the line-definitions: generate sides along the lines that connect surface patches. These sides form an initial front for the triangulation of the surface patches.

F4. Using the information given for the distribution of element size and shape in space, the sides already generated, and the surface definition: triangulate the surfaces. This yields the initial front of faces.

F5. Find the generation parameters (element size, element stretchings and stretching directions) for these faces.

F6. Select the next face to be deleted from the front; in order to avoid large elements crossing over regions of small elements, the face forming the smallest new element is selected as the next face to be deleted from the list of faces.

F7. For the face to be deleted:

F7.1 Select a 'best point' position for the introduction of a new point IPNEW.

F7.2 Determine whether a point exists in the already generated grid that should be used in lieu of the new point. If there is such a point, set this point to IPNEW and continue searching (go to F7.2).

F7.3 Determine whether the element formed with the selected point IPNEW crosses any given faces. If it does, select a new point as IPNEW and try again (go to F7.3).

F8. Add the new element, point and faces to their respective lists.

F9. Find the generation parameters for the new faces from the background grid and the sources.

F10. Delete the known faces from the list of faces.

F11. Add the new faces to the front.

F12. If there are any faces left in the front, go to F6.

A recent thesis by Frykestig [23] gives a good comparison of the different possibilities explored to date for selecting the 'best point position', when to use a given point vs the introduction of a new point, specialized data structures for storing and retrieving mesh data, etc. The complexity of the advancing front algorithm is of $O(N \log(N))$, where N denotes the number of elements. Over the years, optimal data structures have been implemented to realize such a favourable scaling [7–10, 24, 25]. The procedure has been used extensively to grid large-scale complex geometry domains [1–3, 26, 27] and within adaptive remeshing procedures [6, 9, 28–32]. Sustained speeds in excess of 50 000 tetrahedra/min have been achieved on the CRAY-YMP [10, 29].

3. Extending the Range of Applicability

Having outlined the basic advancing front technique, the next logical step is to extend the method's range of applicability. This section treats multimaterial or multidomain applications, volume meshing for thin or crossing surfaces, the generation of grids suitable for Reynolds-averaged Navier–Stokes simulations, higher order elements, and the generation of grids consisting solely of quadrilateral or hexahedral elements.

3.1. Multimaterial or Multidomain Problems

Most large-scale fluid dynamics problems assume a connected region where the material properties (pressures, temperatures, viscosities, etc.) vary according to a single equation of state. The situation is very different in structural mechanics, where multimaterial applications are common. The extension of the

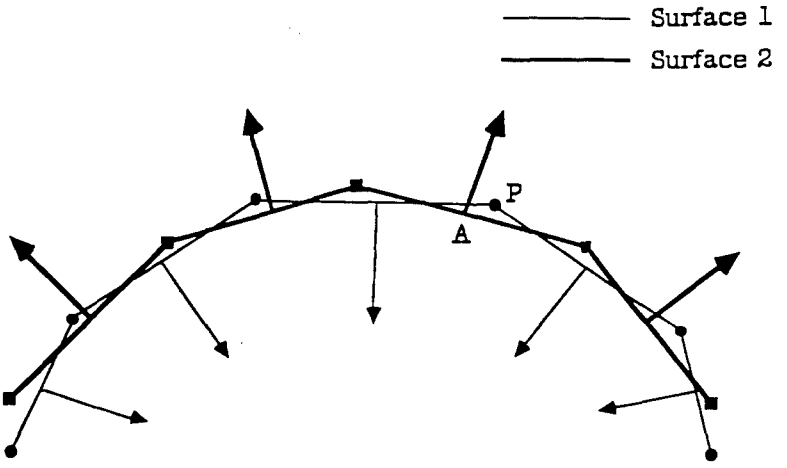


Fig. 1. Crossing of faces for thin surfaces.

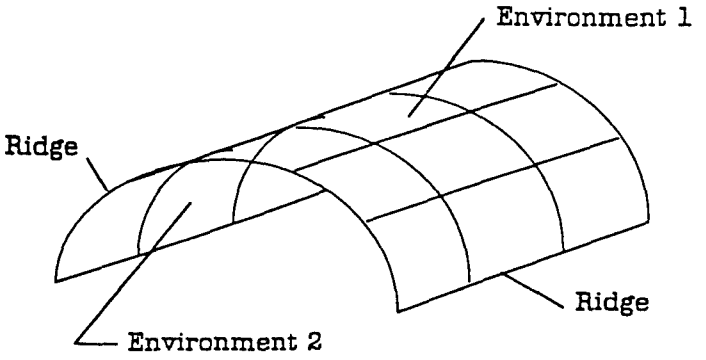


Fig. 2. Environment variable definition.

advancing front technique to these situations is straightforward. In a first pass, all the surfaces defining the inner and outer boundaries of the domains to be gridded are triangulated. Thereafter, a loop is performed over all the domains. For each domain, the faces corresponding to it are assembled and oriented in the proper direction. A tetrahedral mesh is then generated for this domain. A domain identifier is attached to the newly created elements. The next domain is then processed in turn, until all domains have been gridded.

3.2. Volume Meshing for Thin or Crossing Surfaces

In many applications, portions of the boundaries to be gridded will come very close together or even cross. Examples where this occurs are the external meshing of thin-walled structures, such as shells (roofs, walls, etc.), membranes or fabrics (parachutes, sails, parasols, airbags, etc.), or CAD data that exhibit cusps (trailing edges of airfoils and wings). For these cases, the initial surface triangulation will in all probability exhibit crossing faces and/or duplicate points. The application of the usual advancing front technique to this class of problems is not possible, as there is no mechanism to distinguish between the points that may or may not lead to a crossed front. Suppose that face A in Fig. 1 is to be eliminated from the front. Point P will be the point chosen when eliminating this face to form a new element, implying that the 'outside' of the domain to be gridded now contains an element. The occurrence of faces that are extremely close or crossing can very quickly lead to a failure of the advancing front technique, making it impossible to treat these problems on a routine basis. One possible way to circumvent this problem is to mark the faces of the initial front with a so-called *crossing environment variable* LFACR (1:NFACE). This variable can either be obtained by checking the initial front for crossing faces, or by marking the different surface patches that comprise a thin structure, fabric, or are crossing, before the surface grid is generated. In the latter case, the surface faces inherit the crossing environment variable from the surface patch they belong to. As an example, the surfaces shown in Fig. 2 have been marked with such variables. Notice that not every surface patch has its own environment variable, but that all the surfaces lying on one side of a potentially troublesome region have been given the same environment variable. After marking all the surface faces appropriately, the points are marked according to the environment variable of the faces surrounding them. The points that belong

to more than one environment, such as may occur along the ridges that separate two such regions (see Fig. 2), are marked as LPOCR(IPOIN) = −1. The marked points and faces can be used in a variety of ways to make sure that only the proper points and faces are considered when introducing new elements for a face marked as belonging to a particular environment. The two most important are:

(a) No *points* belonging to any other positive environment are considered for the creation of a new element with the face being deleted from the front. This avoids most of the possible logic mistakes that would lead to failure. For the situation shown in Fig. 1, this would eliminate all potentially troublesome points from the list of possible points for face IFOUT, including point P.

(b) No *faces* belonging to any other enviroment are considered for the face-crossing checks. Keeping the faces that belong to the 'other side' of a thin surface situation would render it impossible to introduce new faces on any of the two sides. This is because some of these faces will always be close enough or crossing the newly formed element. For this reason, only the faces belonging to the present environment or no environment (LFACR (IFACE = 0) are kept for front-crossing checks.

New (domain) *points* are always assigned to the value LPOCR(IPOIN) = 0. For the new *faces*, LFACR (IFACE) is set to the maximum value of LPOCR encountered over the three points IP1, IP2, IP3 belonging to it:

LFACR(IFACE) = MAX(0, LPOCR(IP1),
LPOCR(IP2), LPOCR(IP3))

In this way, all faces touching the surfaces marked as belonging to an environment are marked as well. In order to grid as straightforwardly as possible the regions immediately adjacent to troublesome surfaces, the faces marked as belonging to an environment are given the highest priority for deletion from the active front. For advancing front generators that choose the face forming the smallest new element as the one to be deleted next, the marked faces are artificially set to a very small element size. In this way, they are processed first.

3.3. Generation of Grids Suitable for Navier–Stokes Calculations

The task of gridding complex geometries for the simulation of flows using the Navier–Stokes or Reynolds-averaged Navier–Stokes equations (RANS), i.e. including the effects of viscosity and the associated boundary or mixing layers, is encountered commonly in engineering practice. For high Reynolds numbers, the proper discretization of the very thin, yet important boundary or mixing layers requires elements with aspect ratios well in excess of 1:1000. This requirement presents formidable difficulties to general, 'black-box' unstructured grid generators. These difficulties can be grouped into two main categories:

(a) *Amount of Manual Input* In most unstructured grid generators, the desired spatial distribution of element size and shape is given by some form of background grid or sources [6–8, 10]. This seems natural within an adaptive context, as a given grid, combined with a suitable error indicator/estimator, can then be used as a background grid to generate an even better grid for the problem at hand. Consider now trying to generate from manual input a first grid that achieves stretching ratios in excess of 1:1000. The amount of background gridpoints or sources required will be proportional to the curvature of the objects immersed in the flowfield. This implies an enormous amount of manual labour for general geometries, rendering this approach impractical.

(b) *Loss of Control* Most unstructured grid generators introduce a point or element at a time, checking the surrounding neighborhood for compatibilty. These checks involve jacobians of elements and their inverses, distance functions, and other geometrical operations that involve multiple products of coordinate differences. As the stretching ratio increases, round-off errors can become a problem. To see this, consider the mesh around a Boeing-747. The domain length will be approximately 10^3 m, which corresponds to $O(10)$ body lengths. The minimum element length normal to the wing will have to be less than 0.01 mm in order to capture accurately the boundary layer, and 0.05 m in the other two directions. The maximum element length in the farfield will be of the order of 20 m. For a mesh of this kind, the ratio of element volumes is of the order of 3×10^{-12}. Although this is well within reach of the 10^{-16}-accuracy of 64-bit arithmetic, element distortion and surface singularities, as well as loss of control of element shape can quickly push this ratio to the limit.

Given these difficulties, it is not surprising that at present, there does not exist a 'black-box' unstructured (or structured, for that matter) grid generator that can produce acceptable meshes with such high aspect ratio elements. The most common way to generate meshes suitable for Navier–Stokes calculations for complex geometries is to employ a structured or semi-structured

mesh close to wetted surfaces or wakes [33–36]. This 'Navier–Stokes' region mesh is then linked to an outer unstructured grid that covers the 'inviscid' regions. In this way, the geometric complexity is solved using unstructured grids and the physical complexity of near-wall or wake regions is solved by semi-structured grids. This approach has proven very powerful in the past, as evidenced by many examples. A recurring problem in all of these approaches has been how to link the semi-structured mesh region with the unstructured mesh region. Regions where such a link becomes problematic are corners, edges or surfaces with high curvature, as well as regions between surfaces that are very close. In these regions, the elements tend to be either too large or too small, in some cases even folded, and there is usually a sudden jump in element size when entering the unstructured grid region. The design criteria for the grid generation strategy pursued here may be summarized as follows:

—The geometric flexibility of the unstructured grid generator should not be compromised for Navier–Stokes meshes. This implies using unstructured grids for the surface discretization.
—The manual input required for a desired Navier–Stokes mesh should be as low as that used for the Euler case. In the present case, this requirement is solved by specifying at the points of the background grid the boundary layer thickness and the geometric progression normal to the surface.
—The generation of the semi-structured grid should be fast. Experience shows that usually more than half of the elements of a typical Navier–Stokes mesh are located in the boundary-layer regions. This requirement is met by constructing the semi-structured grids with the same normals as encountered on the surface (see Fig. 3), i.e. without recurring to smoothing procedures as the semi-structured mesh is advanced into the field [35, 36].
—The element size and shape should vary smoothly when going from the semi-structured to the fully unstructured mesh regions.
—The grid generation procedure should avoid all of the problems typically associated with the generation of Navier–Stokes meshes for regions with high surface curvature: negative or deformed elements due to converging normals, and elements that get too large due to diverging normals at the surface. In order to circumvent these problems, the same techniques which are used to achieve a smooth matching of semi-structured and unstructured mesh regions are used.

Given these design criteria, as well as the approaches used to meet them, the RANS grid generation algorithm can be summarized as follows (see Fig. 3):

M1. Given a surface definition and a background grid, generate a surface triangulation using an unstructured grid generator.
M2. From the surface triangulation, obtain the surface normals.
M3. Smooth the surface normals in several passes in order to obtain a more uniform mesh in regions with high surface curvature.
M4. Construct a semi-structured grid with the information provided by the background grid and the smoothed normals.
M5. Examine each element in this semi-structured region for size and shape; remove all elements that do not meet certain specified quality criteria.
M6. Examine whether elements in this semi-structured region cross each other; if so, keep the smaller elements and remove the larger ones, until no crossing occurs.
M7. Examine whether elements in this semi-structured region cross boundaries; if so, remove the crossing elements.
M8. Mesh the as yet 'empty' regions of the computational domain using an unstructured grid generator in combination with the desired element size and shape.

Strategies that are similar to the one outlined above have recently been put forward by Pirzadeh [37, 38], Müller [19], and Morgan *et al.* [39], and others. In the following, we consider the main ingredients required for this technique in more detail.

3.3.1. Element Removal Criteria

A critical component of the RANS gridding algorithm described above is the proper matching of semi-structured and fully unstructured grids. This requires good element removal criteria. The criteria to be considered are: element size, element shape, element overlap and element crossing of boundary faces.

(a) *Element Size.* The two main types of problems encountered in semi-structured grid regions that are related to element size are elements that are either too large or negative (folded). These problems originate for different reasons. Elements that are too large may occur if the surface normals diverge close to convex surfaces of high curvature. The situation is shown diagrammatically in Fig. 4. The volume of each element in the semi-structured mesh region is compared to the element volume desired by the user for the particular location in space. Any element with a volume greater than the one specified by the user is

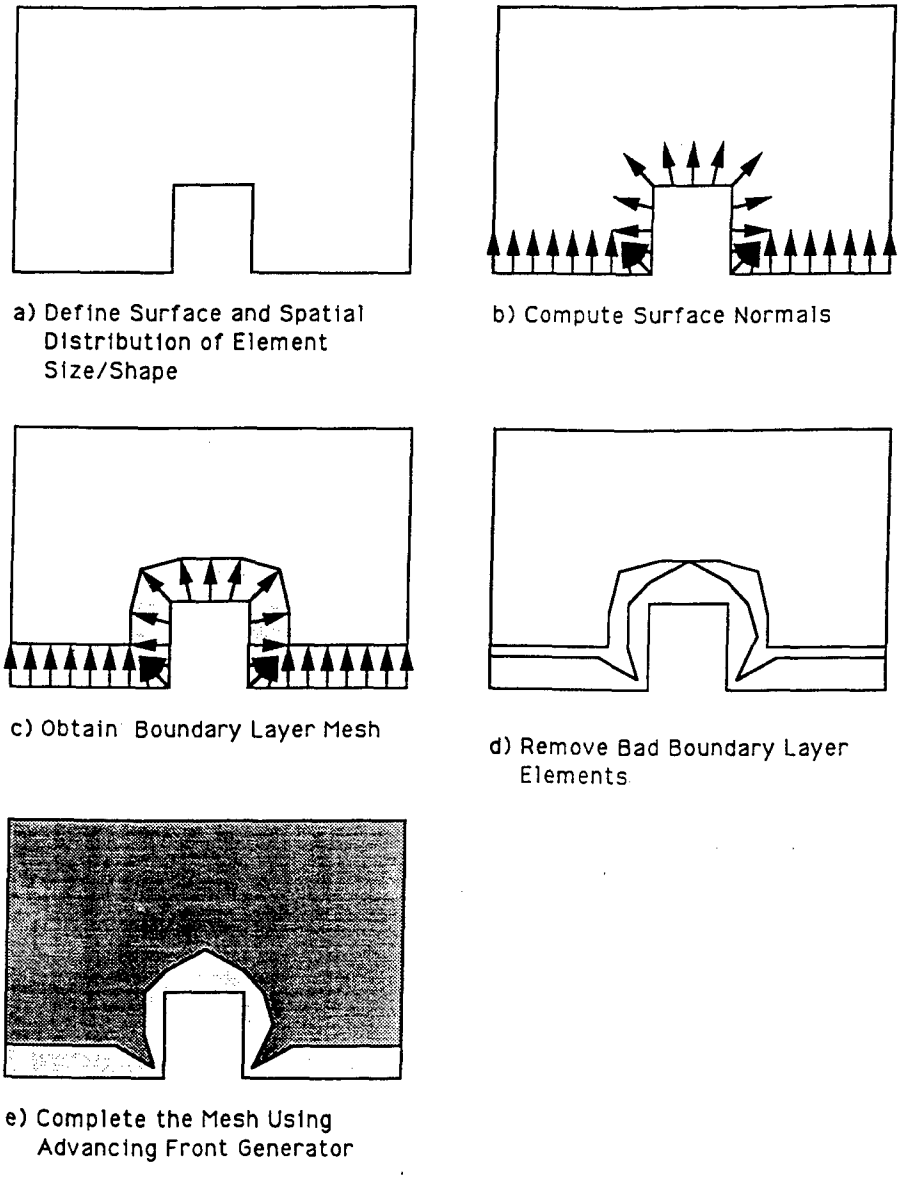


Fig. 3. Navier–Stokes meshing procedure.

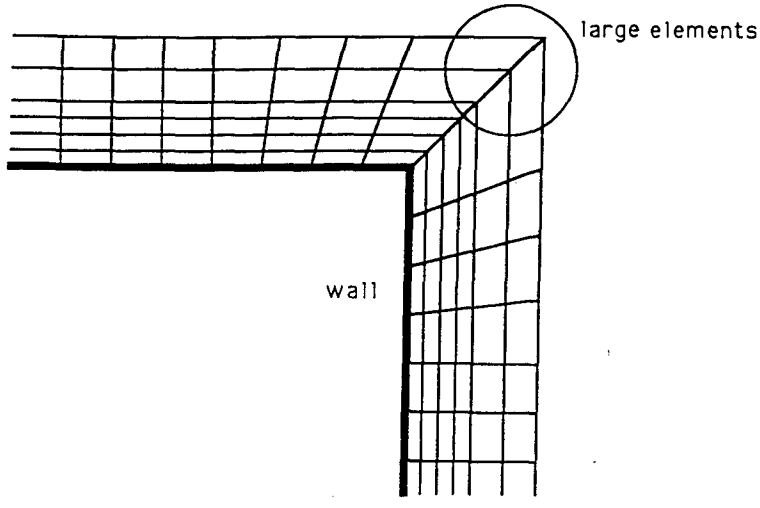


Fig. 4. Element distortion close to a convex corner.

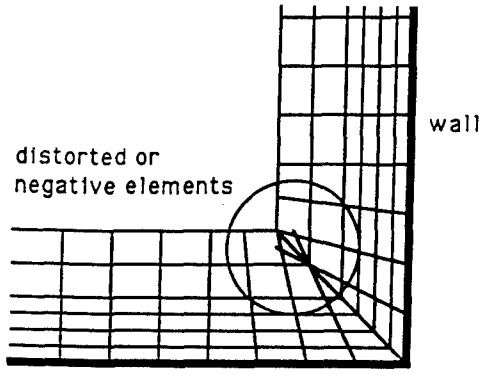


Fig. 5. Element distortion close to a concave corner.

marked for deletion. When concave surfaces exhibit high curvature, the normals will tend to come together or even cross, leading to elements with negative jacobians. The situation is shown diagrammatically in Fig. 5. As before, the element volumes are computed, and elements with negative volumes are marked for deletion. We have observed that typically the elements adjacent to negative elements tend to be highly deformed. Therefore, we also remove all elements that have points in common with negative elements. Obviously, this one-pass procedure can be extended to several passes, i.e. neighbors of neighbors, etc. Our experience indicates, however, that one pass is sufficient for most cases.

(*b*) *Element Shape.* The aim of a semi-structured mesh close to a wall is to provide elements with very small size normal to the wall and reasonable size along the wall. Due to different meshing requirements along the wall (e.g. corners, separation points, leading and trailing edges for small element size, other regions with larger element size), elements that are longer in the direction normal to the wall than along the wall may appear. The occurrence of such elements is shown diagrammatically in Fig. 6. For the semi-structured grids, the element and point numbering can be assumed as known. Therefore a local element analysis can be performed to determine whether side-ratios are consistent with boundary layer gridding. All elements that do not satisfy this criterion are removed.

(*c*) *Overlapping Elements.* Crossing or overlapping elements occur in regions close to concave surfaces with high curvature, or when the semi-structured grids of two close objects overlap. Another possible scenario is the overlap of the semi-structured grids of mixing wakes. The main criterion employed is to keep the smaller element whenever an overlap occurs. In this way, the small elements close to surfaces are always retained. Straightforward testing would result in $O(N_{el})$ operations per element, where N_{el} denotes the number of elements, leading to a total number of operations of $O(N_{el}^2)$. By using quad/octrees [7], or other suitable data structures [23, 24], the number of elements tested can be reduced significantly, leading to a total number of operations of $O(N_{el} \log N_{el})$.

(*d*) *Elements Crossing Boundary Faces.* In regions where the distance between surfaces is very small, the crossing of boundary faces by elements from the semi-structured region is likely to occur. As this test is performed after the element crossing tests are conducted, the only boundaries that need to be treated are those that have no semi-structured grid attached to it. In order to detect if overlapping occurs, we loop over the surface faces, seeing if any element crosses it. As before, straightforward testing would result in an expensive $O(N_{el} \cdot N_f)$ procedure, where N_f denotes the number of boundary faces. Again, the use of quad/octrees reduces the complexity to $O(N_f \log N_{el})$.

3.3.2. Smoothing of Surface Normals

Smoothing of surface normals is always advisable for regions with high surface curvature, particularly corners, ridges and intersections. In order to start the smoothing process, initial point-normals RNORO, as well as boundary conditions for point-normals must be provided. The normal at any point is computed by

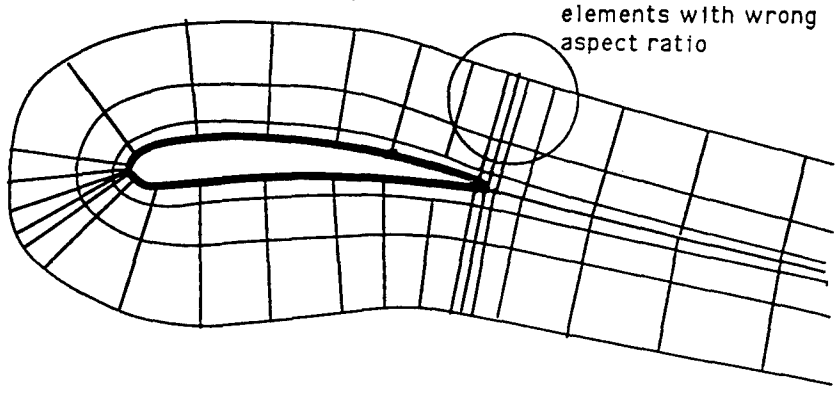


Fig. 6. Elements with undesired shape.

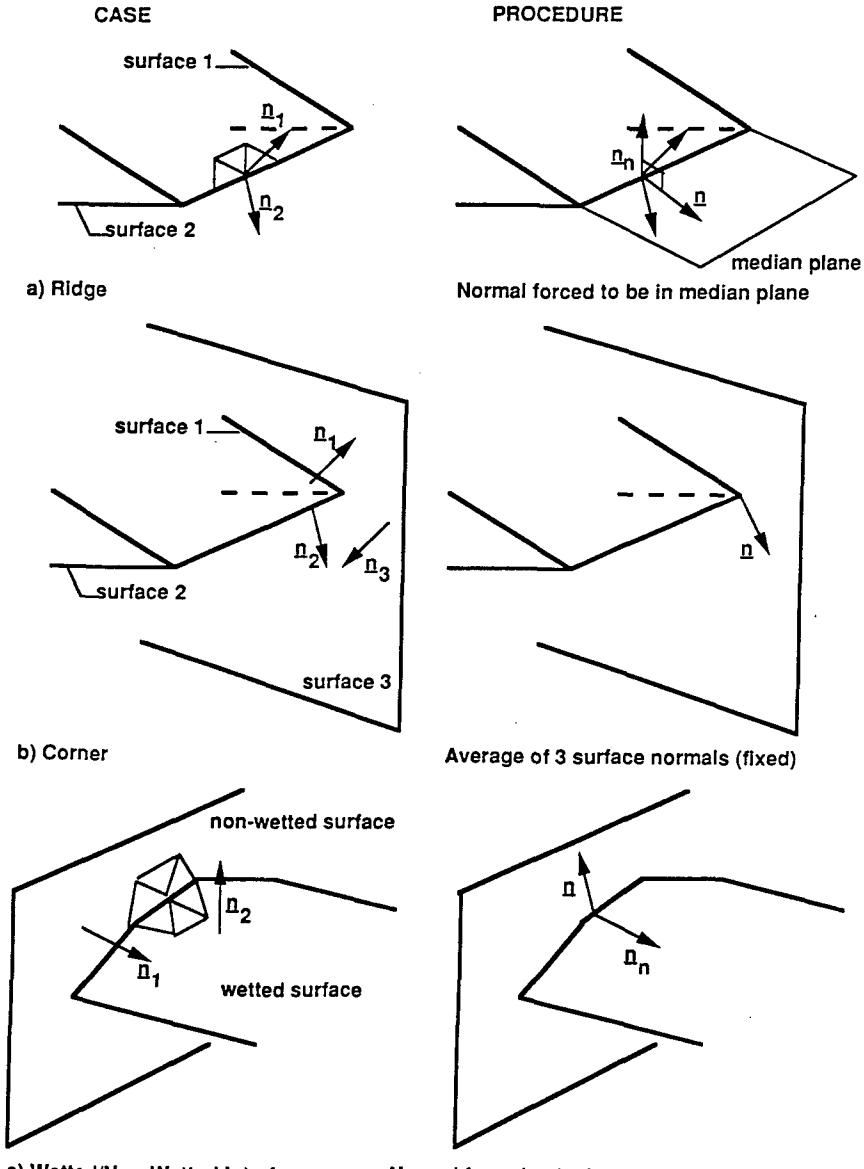


Fig. 7. Boundary conditions for normals.

averaging the normals of all faces touching it. The choice of boundary conditions is crucial in order to ensure that no negative elements are produced at corners, ridges and intersections. Figure 7 shows a number of possibilities. Note that the trailing edge of wings (a particularly difficult case requiring particular attention) falls under one of these categories. In each smoothing pass, the point-normals are averaged in the faces. Thereafter, the new face-normals are assembled additively at the points, and normalized. The complete smoothing procedure may require in excess of 200 passes in order to converge. This slow convergence may be speeded up considerably through the use of conjugate gradient [40] or superstep [41, 42] acceleration procedures. Employing the latter procedure, convergence is usually achieved in less than 20 passes.

3.3.3. Point Distribution along Normals

Ideally, we would prefer to have normals that are perpendicular to the surface in the immediate vicinity of the body, and smoothed normals away from the surface [43]. Such a blend may be accomplished by using Hermitian polynomials. If we assume given:

—the points of surface $\mathbf{x}_0$;
—the boundary layer thickness δ;
—the surface normals before and after smoothing $\mathbf{n}_0, \mathbf{n}_1$;
—a non-dimensional boundary layer point-distribution parameter ξ of the form: $\xi_{i+1} = \alpha\xi_i$, $\xi_n = 1$;

then the following Hermitian cubic polynomial in ξ will produce the desired effect:

$$\mathbf{x} = \mathbf{x}_0 + \xi\delta\mathbf{n}_0 + \xi\cdot(2-\xi)\cdot\xi\delta(\mathbf{n}_1 - \mathbf{n}_0) \qquad (1)$$

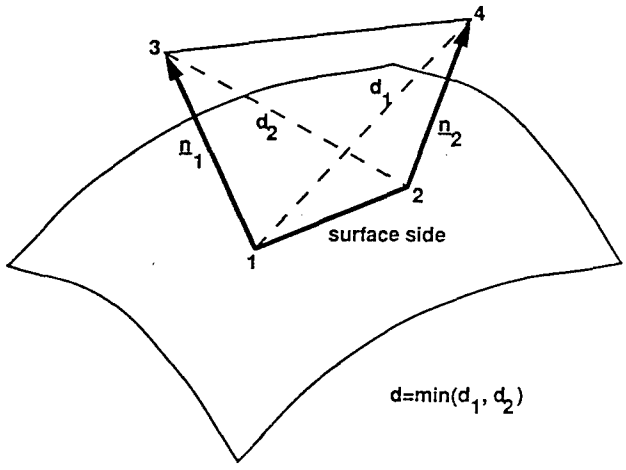


Fig. 8. Selection of diagonals when splitting prisms into tetrahedra.

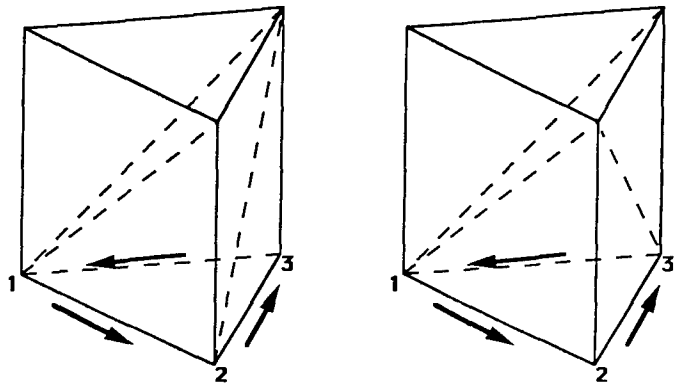


Fig. 9. Splitting of prisms into tetrahedra.

One can readily identify the linear parts $\xi \delta \mathbf{n}_i$. In some cases, a more pronounced (and beneficial) effect may be produced by substituting for the higher-order polynomials a new variable η, i.e.

$$\xi \cdot (2 - \xi) :\to \eta \cdot (2 - \eta), \quad \eta = \xi^p \tag{2}$$

where, e.g. $p = 0.5$.

3.3.4. Subdivision of Prisms into Tetrahedra

One of the aims of the RANS gridding technique is to arrive at a single, smoothly varying mesh consisting of tetrahedral elements only. Therefore, the prisms formed by extruding the surface triangles along the smoothed normals must be subdivided. This subdivision has to be performed in such a way that the diagonals introduced at the rectangular faces of the prisms match across prisms. The best possible diagonal for each of these rectangular faces is chosen in order to avoid large internal angles, as shown in Fig. 8. As this choice is only dependent on the coordinates and normals of the two end-points of a surface side, compatibility across faces is assured. The problem is, however, that a prism cannot be subdivided into tetrahedra in any arbitrary way. Therefore, care has to be taken when choosing these diagonals. Figure 9 illustrates the possible diagonals as the base sides of the prism are traversed. One can see that in order to obtain a combination of diagonals that can be subdivided into tetrahedra, not all sides of the triangular base have to be up–down or down-up as one traverses the sides. This implies that the sides of the triangular base mesh have to be marked in such a way that no such combination occurs. A proper set of diagonals can be produced by a simple iterative procedure described in [44]. When inverting the side-diagonal orientation, those diagonals with the smallest large internal angle are sampled first in order to minimize the appearance of bad elements. Our experience indicates that even for large surface grids (>100 Ktria), the number of diagonals that require inversion of side/diagonal orientation is very small (<15).

3.4. Extension to Higher Order Triangles and Tetrahedra

Once a grid of linear triangles or tetrahedra has been obtained, it may be post-processed further in order to obtain higher order elements. This is particularly useful for structural mechanics problems, where linear triangles and tetrahedra have been found to be notoriously 'stiff' elements [45]. Quadratic triangles and tetrahedra offer an interesting alternative for complex geometries, and have gained widespread use and acceptance in recent years. For quadratic elements, all that is required is the introduction of new points along the edges (or sides) of the original linear element mesh. These new points are then introduced in the connectivity matrix. Finally, the positions of the new boundary points are corrected for curved boundaries. Cubic or quartic elements require some more work, as the new degrees of freedom can now lie on element faces, or within the elements.

3.5. Extension to Quadrilateral and Hexahedral Elements

The structural mechanics community has had a historic preference for quadrilateral or hexahedral elements. High-speed structural dynamics codes, such as DYNA3D or NIKE3D [45, 47], have used these types of elements almost exclusively. The generation of quadrilateral elements on any 3-D surface may either be accomplished *ab initio* using paving techniques [11], or by first generating a mesh of triangles that is then post-processed further. The second approach proceeds through the following steps:

(a) generate a mesh of triangles with element size twice as large as the final desired quadrilateral element mesh;
(b) join as many triangles as possible into well-shaped quadrilaterals;
(c) smooth the grid of triangles and quadrilaterals obtained in the previous step;
(d) subdivide the elements further: triangles are converted into three new quadrilaterals, while quadrilaterals yield four new quadrilaterals;
(e) smooth the final all quadrilateral mesh.

The generation of hexahedral elements is a different matter entirely. If the starting point is a grid of tetrahedra, two ways to proceed are possible: (a) fusion of tetrahedra into other polyhedra, or (b) fission of tetrahedra into hexahedral elements.

We have tried both approaches. For the fusion technique, the aim is to join tetrahedra into other polyhedra (pyramids, prisms, hexahedra), in order to arrive at grids that contain a minimum amount of tetrahedra. This effort was not successful. The problem is the proper transition of elements with quad-faces to elements with triangular faces. Even though all combinations were tried, it was found that the 'yield' of good hexahedra was below 5% for typical large tetrahedra grids. The second approach is to subdivide each tetrahedron into four hexahedra. While the quality of such a hexahedra-only grid may appear questionable, we have found no appreciable degradation by using them within DYNA3D [48].

4. Improvements in Speed

One of our aims is to be able to generate grids in excess of a million elements in a matter of minutes on a workstation. For this reason, grid generation speed has been a major focus of research. The incorporation of optimal search procedures, such as octrees and linked lists, has enabled advancing front generators to realize an operational complexity of $O(N \log N)$ [7, 9]. Further reductions in meshing times may be achieved by avoidance of unnecessary search overheads, global h-refinement, vectorization and parallelization.

4.1. Reduction of Search Overheads

Our experience indicates that the number of close points and faces found for checking purposes is far too high. As an example, consider the search for close points: there may be up to eight points inside an octant, but of these only one may be close to the face to be taken out. The idea is to filter out these 'distant' faces and points in order to avoid extra work afterwards. While the search operations are difficult to vectorize, these filtering operations lend themselves to vectorization in a straightforward way, leading to a considerable overall reduction in CPU requirements. Moreover, filtering requires a modest amount of operations as compared to the work required in subsequent operations. The most important filtering operations are:

—removal of points that are too far away;
—removal of points that are not visible from the face to be removed from the front (these would form elements with negative jacobians, see Fig. 10);
—removal of points that are not visible from the visible adjacent faces to the face to be removed from the front (these would form elements that cross the front, see Fig. 11);

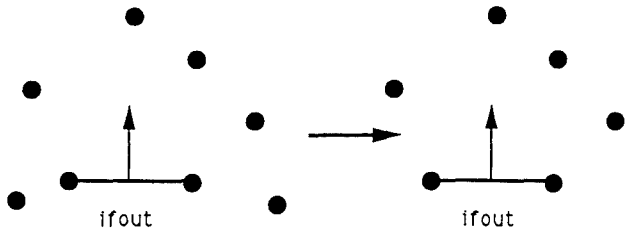


Fig. 10. Removal of points not visible from the face to be removed.

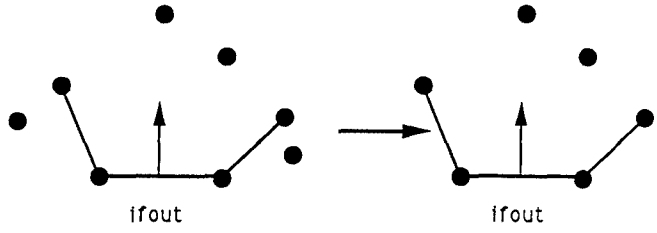


Fig. 11. Removal of points not visible from the visible adjacent faces.

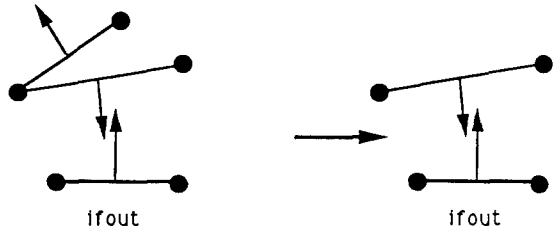


Fig. 12. Removal of faces that cannot see the face to be removed.

—removal of faces that cannot see the face to be removed (there is no need to check for these, see Fig. 12).

4.2. Global h-refinement

While the basic advancing front algorithm is a scalar algorithm, h-refinement can be completely vectorized. Therefore, the grid generation process can be made considerably faster by first generating a coarser mesh that has all the desired variations of element size and shape in space, and then refining globally this first mesh with classic h-refinement [10]. Typical speed-ups achieved by using this approach are 1:6 to 1:7 for each level of global h-refinement. Typical 3-D advancing front generators construct grids at a rate of 12 000 tetrahedra per minute on the IBM-RISC/550 and 50 000 tetrahedra per minute on the CRAY-YMP. With one level of h-refinement, this last rate is boosted to 190 000 tetrahedra per minute. This rate is essentially independent of grid-size, but can decrease for very small grids.

4.3. Vectorization

When generating large grids on traditional vector-supercomputers, due care has to be taken to achieve a high degree of vectorization. The advancing front technique is a sequential procedure, in which elements are introduced one at a time. This would seem to preclude efficient vectorization. However, many of the time-consuming search operations, such as the face-crossing and filtering techniques, may be readily vectorized. The author's advancing front generator achieves a speed-up of 1:6 on the CRAY-YMP when vectorization is enabled, indicating a fair degree of vectorization.

4.4. Parallelization

As seen before, the advancing front technique is essentially a scalar technique as far as the introduction of elements is concerned. However, given a sufficient distance between them, many elements may be introduced at the same time. Given that the number of operations required for the creation of a new element can vary by orders of magnitude (very small for a face on a convex front far away from any other front faces, large in regions of collapsing fronts), only MIMD machines can be contemplated for parallelization [49, 50]. The background grid provides a useful means to partition the domain to be gridded into subdomains. Elements are generated in parallel in each of these subregions. Thereafter, the regions between the subdomains are also generated in parallel. One processor, which acts as master, directs the gridding of the subregions by allocating and transmitting to each of the other 'slave' processors the active faces and points of the front, as well as the background grid elements corresponding to this subregion. Once each of the other processors has completed the meshing of its subregions, the information is transmitted back to the master processor. Given that the time required to generate a tetrahedral mesh is several orders of magnitude larger than the transmission times for the information required for it, this 'card-dealer' hierarchy of processors works well for up to several hundred processors. For parallel machines with an even higher number of processors, intermediate 'master-slave' hierarchies can be employed. As the number or processors grows beyond ten, it becomes impossible to store the complete grid in a single processor. In order to circumvent this limitation, a '*distributed id*' data structure was introduced in [50].

Given that the typical user prefers coarse background grids, finer background grids that allow for well-balanced subdivisions with uniform work loads

across them are generated via a pre-processor. The details of this procedure, as well as further enhancements of the parallel grid generation procedure may be found in [50].

5. Improvement in User-Friendliness

Improvements in range of capabilities and speed would be of little use unless similar enhancements were achieved in user-friendliness. In this section, we treat two areas where user-friendliness has been improved significantly: surface definition from discrete data, and the reduction of manual input to specify arbitrary element size and shape distribution in space.

5.1. Surface Definition from Discrete Data

For complex configurations, the most man-hour intensive part of the analysis process is the preparation of the surface data. Several reasons can be given for this:

(a) The amount of CAD data can be very large. Consider the complete external shape of an airplane, car or ship, the cooling head of a reciprocating engine block, or a complex stamping part. In all of these cases, the analyst will be faced with hundreds, if not thousands of surface patches that have to be merged, coordinated, and possibly intersected and trimmed. This is a tedious and thankless task that demands many man-hours.
(b) The data used by one department of a company may not even be compatible with the data used by another department of the same company, let alone those used by other companies. Even after years of effort to standardize CAD input and output formats, in practice the data sets stemming from different CAD packages are not compatible. In the United States in particular, many large companies still have their own in-house CAD system, which is incompatible with that of other companies or the IGES standard.
(c) Even if the data can be read by another CAD package, the splines, b-splines or other means of defining the geometry may not be the same as that of the originating package, prompting errors, corrections, delays, and the associated increase in analysis cost and uncertainty.

One possible solution to this dilemma is to do away with CAD data in the form of continuous functions, and to start from a cloud of points that defines the surface of the domain to be gridded. This first line of thought may be summarized as follows:

(a) Any grid required for analysis does not consist of infinitesimally small elements, but of a finite amount of (finite) elements.
(b) A point lying on the surface is the only unique item that can define unequivocally the surface of the domain to be gridded (support points and the functions used would do so too: the problem is that in practice, we are only provided with the support points, but not the functions).
(c) A point lying on the surface can easily be obtained from any CAD-system.
(d) If the cloud of points is dense enough, whenever a point is required for the grid used for analysis, it can be obtained from the cloud of points.

A second line of thought is prompted by the observation that in many instances, the CAD model is either not available, or is not the starting point for the surface description. Examples for surface information obtained discretely *ab initio* include: terrain data gathered via remote sensing for climate predictions, geological data gathered via seismic analysis or probing for seepage and ground-water flow simulations, digitization of clay models for the design of cars and motorcycles, and CAT-scan data for biomedical applications. In any of these cases, the surface is defined by a point-set.

Surface reconstruction from clouds of points has been pursued for many years by computer scientists. In some instances, remarkably complex structures have been reconstructed [51–53].

The advancing front technique may also be used to construct a surface triangulation from the cloud of points. Starting from a point in the domain, an initial side is built with the nearest neighbor. Thereafter, a third point in the neighborhood is selected to construct the first triangle. We now have an initial front of three sides, as well as a surface normal associated with each side given by the normal to the triangle. The sides and surface normal are oriented in such a way that the normal is pointing in the direction of the reference points. If the three reference points cannot be seen by the present face and its normal, another set of three points is selected. This initial face and its associated sides define the initial front. We then proceed, for each subsequent side, as follows:

T1. Select the smallest side of the current front as the next side to be deleted from the front.
T2. Obtain all the points in the neighborhood of this side.

T3. Filter out the points that are not within the 'cone of visibility' of the side.
T4. IF: no points are left:
—mark the side as an edge-side;
—add it to the bottom of the side-list;
—proceed to the next side (GOTO T.1).
ELSE:
—obtain all the sides in the neighborhood;
—order the points according to suitability;
—select the 'best' close point that does not cross any of the current front-sides to form a new triangular face;
—add the new face to the surface grid;
—compute the surface normal for the new face;
—update all front arrays;
ENDIF
T5. If any sides remain in the front: GOTO T.1.

This procedure works well for clouds of points that are sufficiently fine [54]. We have found that in order to avoid problems of non-uniqueness at ridges and corners, it is important to start the triangulation procedure using as the initial point the one for which the surface curvature is minimal. Procedures to define surface curvature for a discrete set of points, as well as further refinements may be found in [54].

5.2. Reduction of Input Requirements to Specify Element Size

The specification of the desired element size and shape in space has been a recurring problem for most grid generators. This is because the requirements for simplicity (low user input) and flexibility (complex geometries) are conflicting. The earliest advancing front grid generators employed a background grid to specify the desired element size and shape in space. This worked well for simple geometries, and was particularly suited for adaptive remeshing procedures [6, 28, 29]. For CAD-based surface descriptions, the modified or finite quad- and octree techniques provide an automatic way of refining the mesh in regions of high surface curvature [20, 21]. This works well for problems that require a fine mesh in regions of high surface curvature, and a coarser mesh away from surfaces. While this is indeed the case for many elliptic problems, a user may still wish to refine the mesh in some arbitrary spatial region of space (e.g. a heat-source, an oblique shock in supersonic flow, etc.). Therefore, alternative ways to prescribe element size and shape in space, that combine generality and low user input, are required.

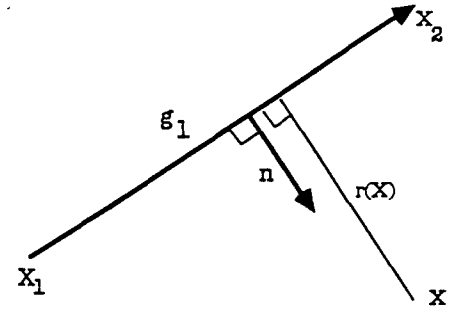


Fig. 13. Line source.

5.2.1. Sources

A flexible way that combines low user input, arbitrary but smoothly varying element size, and can be associated with CAD data is to define sources. The element size for an arbitrary location $\mathbf{x}$ in space is given as a function of the closest distance to the source $r(\mathbf{x})$. Consider first the line source given by the points $\mathbf{x}_1, \mathbf{x}_2$ shown in Fig. 13. The vector $\mathbf{x}$ can be decomposed into a portion lying along the line, and the normal to it. With the notation of Fig. 13, we have

$$\mathbf{x} = \mathbf{x}_1 + \xi \mathbf{g}_1 + \alpha \mathbf{n} \tag{3}$$

The ξ can be obtained by scalar multiplication with $\mathbf{g}_1$, and is given by:

$$\xi = \frac{(\mathbf{x} - \mathbf{x}_1)\cdot \mathbf{g}_1}{\mathbf{g}_1 \cdot \mathbf{g}_1} \tag{4}$$

By delimiting the value of ξ to be on the line:

$$\xi' = \max(0, \min(1, \xi)) \tag{5}$$

the distance between the point $\mathbf{x}$ and the closest point on the line source is given by:

$$\delta(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_1 - \xi' \mathbf{g}_1| \tag{6}$$

Point sources can be constructed by collapsing the line end-points into one. Consider next the surface source element given by the points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ shown in Fig. 14. The vector $\mathbf{x}$ can be decomposed into a portion lying in the plane given by the surface source points, and the normal to it. With the notation of Fig. 14, we have

$$\mathbf{x} = \mathbf{x}_1 + \xi \mathbf{g}_1 + \eta \mathbf{g}_2 + \gamma \mathbf{g}_3 \tag{7}$$

where

$$\mathbf{g}_3 = \frac{\mathbf{g}_1 \times \mathbf{g}_2}{|\mathbf{g}_1 \times \mathbf{g}_2|} \tag{8}$$

By using the contravariant vectors $\mathbf{g}^1, \mathbf{g}^2$, where $\mathbf{g}_i \cdot \mathbf{g}^j = \delta_i^j$, we have

$$\xi = (\mathbf{x} - \mathbf{x}_1)\cdot \mathbf{g}^1, \quad \eta = (\mathbf{x} - \mathbf{x}_1)\cdot \mathbf{g}^2, \quad \zeta = 1 - \xi - \eta \tag{9}$$

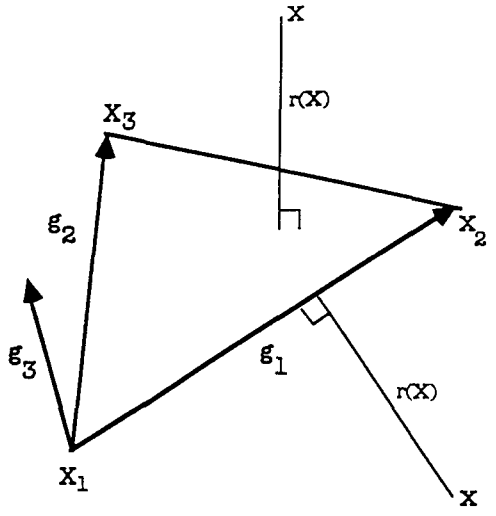


Fig. 14. Surface source.

The point **x** lies 'on the surface' if:

$$0 \leq \xi, \eta, \zeta \leq 1 \tag{10}$$

Whenever this condition is violated, the point **x** will be closest to one of the edges, and the distance to the surface is evaluated by checking the equivalent line sources associated with the edges. If, on the other hand, eqn (10) is satisfied, the closest distance between the surface and the point is given by:

$$\delta(\mathbf{x}) = |(1 - \xi - \eta)\mathbf{x}_1 + \xi\mathbf{x}_2 + \eta\mathbf{x}_3 - \mathbf{x}| \tag{11}$$

As one can see, the number of operations required to determine $\delta(\mathbf{x})$ is not considerable as long as one pre-computes and stores the geometrical parameters of the sources ($\mathbf{g}_i$, $\mathbf{g}^i$, etc.). In order to reduce the internal complexity of a code, it is advisable to only work with one type of source. Given that the most general source is the surface source, line and point sources are prescribed as surface sources, leaving a small distance between the points to avoid numerical problems (e.g. divisions by zero).

Having defined the distance from the source, the next step is to select a function that is general yet requires a minimum amount of input to define the element size as a function of distance. Typically, the user desires a small element size close to the source, and a large element size away from it. Moreover, the element size should, in many instances, be constant (and small) in the vicinity $r < r_0$ of the source. Typical cases that fall under this category are wings and slender bodies or pipes for fluid flow problems. An elegant way to satisfy these requirements is to work with functions of the transformed variable

$$\rho = \max\left(0, \frac{r(\mathbf{x}) - r_0}{r_1}\right) \tag{12}$$

For obvious reasons, the parameter r_1 is called the scaling length. Commonly used functions of ρ used to define the element size in space are:

(a) *Power laws:* given by the expressions of the form [10]

$$\delta(\mathbf{x}) = \delta_0[1 + \rho^\gamma] \tag{13}$$

with the four input parameters $\delta_0, r_0, r_1, \gamma$; typically, $1.0 \leq \gamma \leq 2.0$.

(*b*) *Exponential functions:* which are of the form [18]

$$\delta(\mathbf{x}) = \delta_0 \, e^{\gamma\rho} \tag{14}$$

with the four parameters $\delta_0, r_0, r_1, \gamma$.

(*c*) *Polynomial expressions:* which avoid the high cost of exponents and logarithms by employing expressions of the form:

$$\delta(\mathbf{x}) = \delta_0\left[1 + \sum_{i=1}^{n} a_i\rho^i\right], \tag{15}$$

with the $n + 3$ parameters δ_0, r_0, r_1, a_i. We have found that in practice quadratic polynomials are sufficient, i.e. $n = 2$.

Give a set of m sources, the minimum element size computed for each of them is taken whenever an element is to be generated:

$$\delta(\mathbf{x}) = \min(\delta_1, \delta_1, \ldots, \delta_m) \tag{16}$$

Sources offer a convenient and general way to define the desired element size in space. They can be introduced interactively on a workstation with a mouse-driven menu once the surface data is available. They suffer from one major disadvantage: at every instance, the generation parameters of *all* sources need to be evaluated. For a distance distribution given by eqns (12)–(16), it is very difficult to 'localize' the sources in space in order to filter out the relevant ones. On the other hand, the evaluation of the minimum distance obtained over the sources may be vectorized in a straightforward way. Nevertheless, a high number of sources ($N_s > 100$) will have a marked impact on CPU times, even on a vector-machine.

5.2.2. Element Size Attached to CAD DATA

For problems that require gridding complex geometries, the specification of proper element sizes can become a tedious process. Conventional background grids would involve many tetrahedra, whose generation is a labour-intensive, tedious task. Point, line, or surface sources are not always appropriate either. Curved 'ridges' between surface patches, as sketched in Fig. 15, may require many line sources. Similarly, the specification of gridding parameters for surfaces with high curvature may require many surface sources.

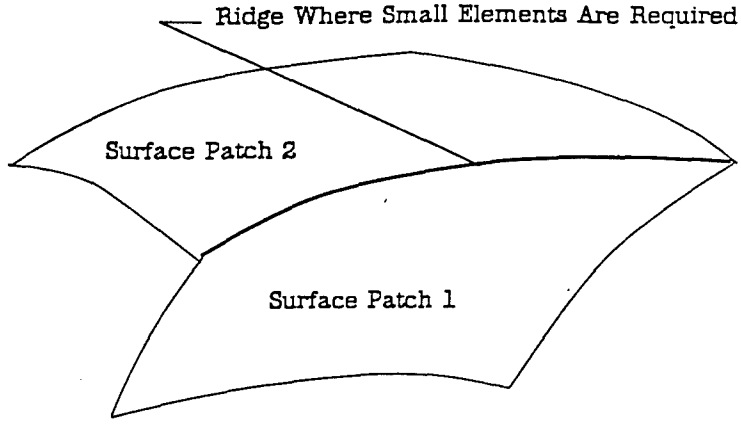


Fig. 15. Element size attached to CAD data.

The net effect is that for complex geometries one is faced with excessive labour costs (background grids, many sources), and/or CPU requirements during mesh generation (many sources).

A better way to address these problems is to attach element size (or other gridding parameters) directly to CAD data. For many problems, the smallest elements are required close to the boundary. Therefore, if the element size for the points of the current front is stored, the next element size may be obtained by multiplying it by a user-specified increase factor c_i. The element size for each new point introduced is then taken as the minimum obtained from the background grid δ_{bg}, the sources δ_s, and the minimum of the point sizes corresponding to the face being deleted, multiplied by a user-specified increase factor c_i:

$$\delta = \min(\delta_{bg}, \delta_s, c_i \cdot \min(\delta_A, \delta_B, \delta_C)) \qquad (17)$$

Typical values for c_i are $1.0 \leq c_i \leq 2.0$. The first value yields a mesh of uniform element size, whereas the latter gives rise to grids with elements that grow rapidly in size away from the surface. Specifying or attaching element sizes to CAD data can lead to incompatibilities if surfaces are close to each other. For example, in the situation shown in Fig. 16,

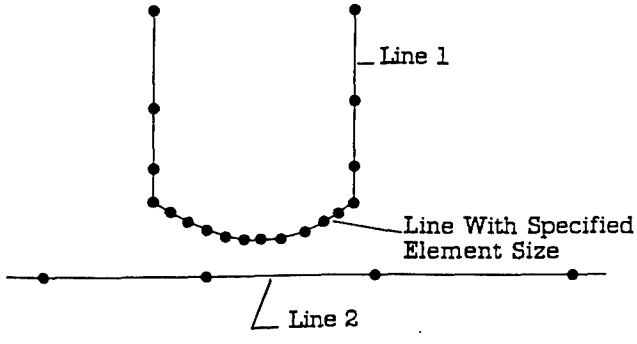


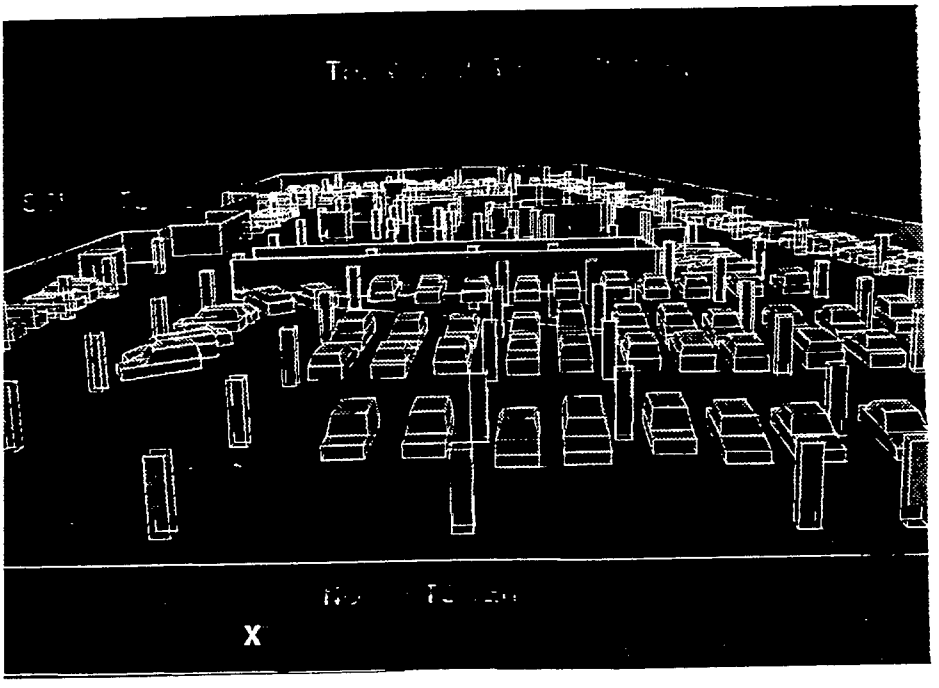
Fig. 16. Potential problems with incompatible sizes.

specifying a small distance for line L1 without proper modification of the distance parameter for line L2 can lead to size incompatibilities and badly shaped elements. This is because this type of specification of element size is 'hyperbolic' by nature, starting from the surfaces and marching blindly into the domain.

6. Examples

6.1. World Trade Center

In order to show the current use of large-scale tetrahedral grids for complex geometries we consider one of the floors of the World Trade Center. This particular configuration included approximately 280 cars, in excess of 100 columns, ramps, inner walls, etc. Some of the data were merged from AutoCad (e.g. columns, ramps), some from architectural drawings (e.g. parking positions, walls), and some from measurements (e.g. cars, beam widths). What is typical for real-life applications of this kind is the convergence of data from several sources (CAD, drawings, models, measurements) that the CFD user must merge and 'trim'. Note that CAD tools by themselves will not solve this task. Rather, specialized software tools that can read CAD data, have some CAD-like functionality to merge and construct data, and impose the proper boundary conditions are required. The preparation of the data for this configuration required 1 week. The surface definition data is shown in Fig. 17(a). The grid consisted of approximately 18 Mtet elements, with 600 ktria surface faces. Although this number of elements at first sight would appear as excessively high, it was the minimum required to reproduce faithfully the geometry and physics under consideration. A simulation was carried out to assess the damage due to blasts. A snapshot of one such run is shown in Fig. 17(b).



(a)

(b)

Fig. 17. World Trade Center B2 Level: (a) surface definition (600 ktria); (b) blast simulation (pressure contours).

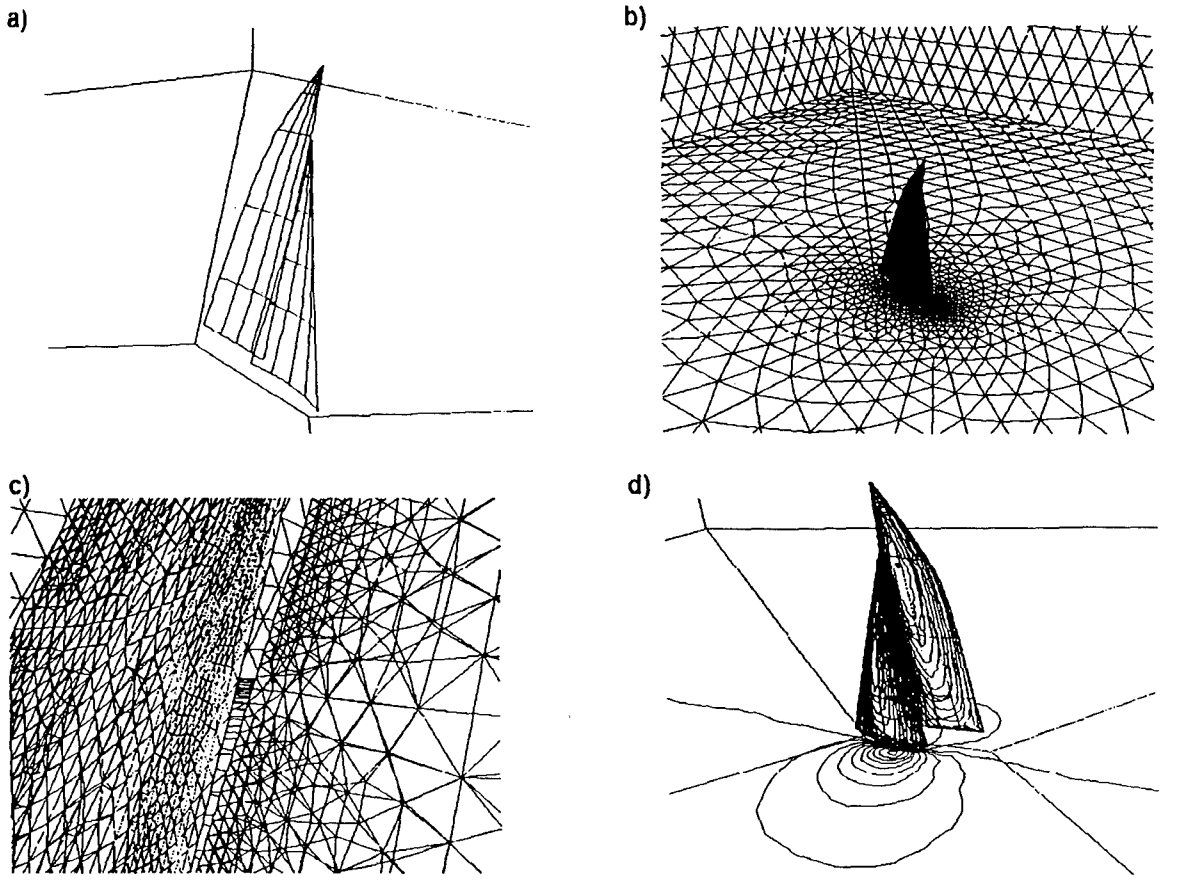


Fig. 18. Sails: (a) surface definition; (b) surface mesh; (c) surface mesh (close-up); (d) surface pressure.

6.2. Sails

This example illustrates an application that requires a volumetric grid around thin or crossing surfaces. The configuration is shown in Fig. 18(a). Note the inclination of the sails, as well as the very narrow gap between the front and back sails. The sails, which measure approximately 30 m, were modeled as having a thickness of 1 mm. For all practical purposes, this is equivalent to a vanishing thickess. A background grid of only five elements (a hexahedron split into tetrahedra) enclosing the domain was used to prescribe linear variation of element size in the vertical direction, with $\delta_0 = 10$ m and $\delta_1 = 30$ m. In addition, two line sources and six surface sources were employed to concentrate smaller elements on the sails and between them. Furthermore, element size attached to CAD data was used for the sails. Two views of the surface mesh are shown in Figs 18(b, c). As before, the surfaces of the sails cross. The final mesh consisted of NELEM = 200 277 elements and NPOIN = 20 383 points. The contours of the surface pressures obtained on this mesh for an inviscid flow simulation are shown in Fig. 18(d).

6.3. Chip

In this particular instance, the proper discretization of the solid domain inside a chip, as well as the surrounding fluid medium, was required. The definition of the surfaces is shown in Fig. 19(a). The complete domain had five different materials, including the fluid. In this instance, no sources were specified. Instead, the option to attach element size directly to CAD data was used extensively. Most of the chip surfaces had an element size attached to them. For the fluid region, a Navier–Stokes grid was required. The surface of the grid, which consisted of approximately 1.5 Mtet elements, is shown in Fig. 19(b). Results from an incompressible flow solution with conjugate heat transfer at a Reynolds number of $Re = 1000$ based on the diameter of the chip are included in Fig. 19(c) to show that these grids can indeed be used for industrial applications. Due to the ability to attach grid-size directly to CAD data, the pre-processing time was kept to a minimum. Less than 4 h (one morning) were required to input all the surface defining information, the grid generation parameters, and to obtain the mesh.

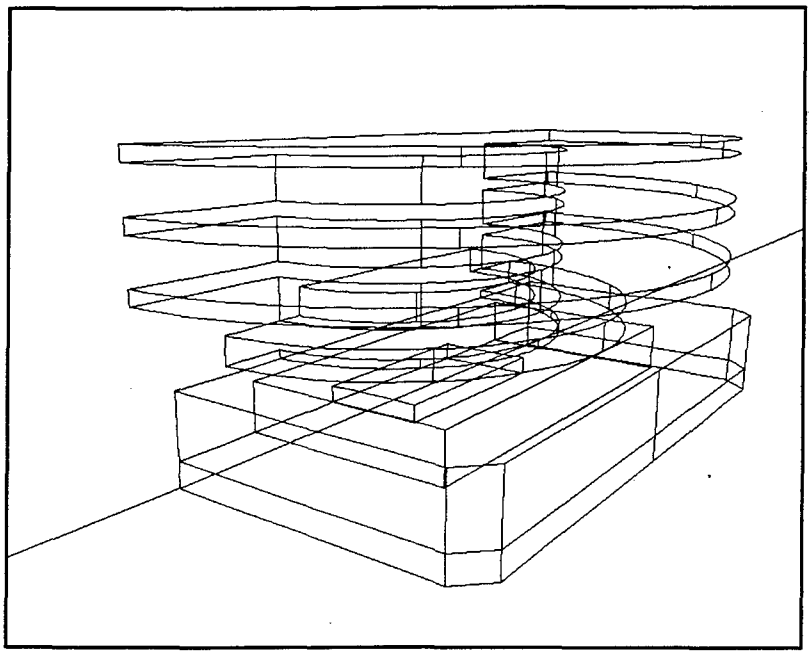


Fig. 19. (a).

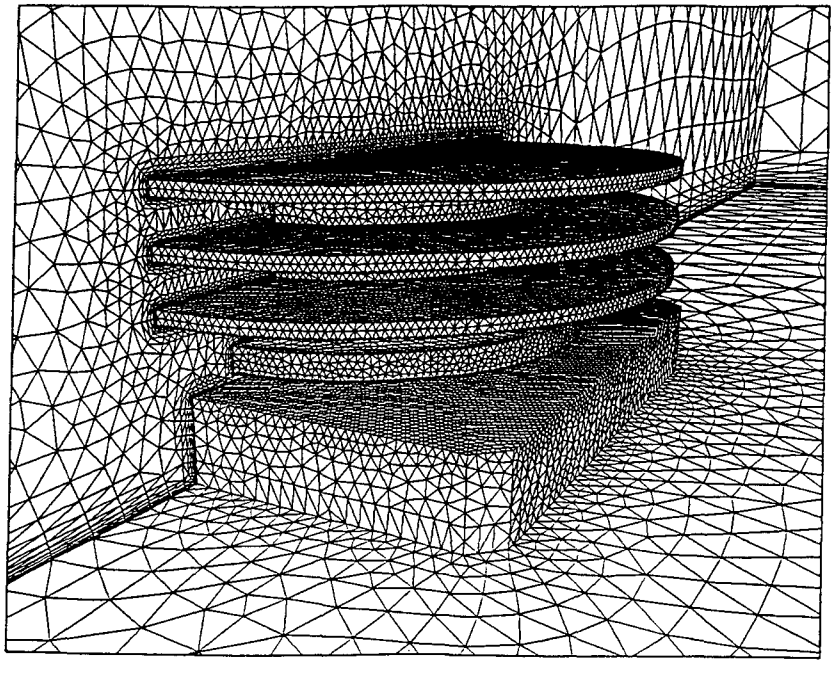
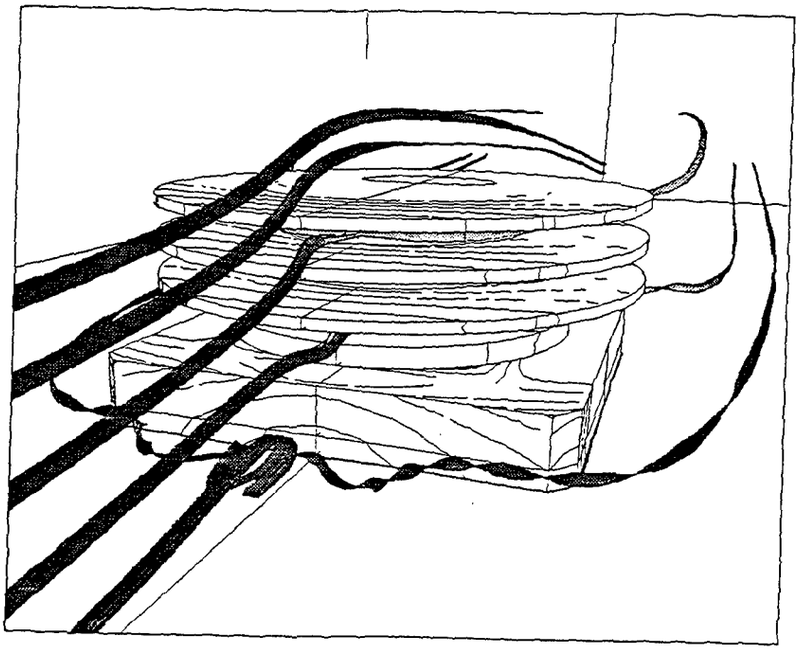


Fig. 19. (b).

6.4. Submarine Surface

For this case, an all-quad surface discretization for the hull of a generic submarine configuration was required for structural mechanics applications. The surface definition is shown in Fig. 20(a). A total of 24 line sources were used to specify the desired element size in space. An initial surface grid with approximately 24 000 triangular elements was built. Thereafter, as many elements as possible were fused into 'good' quadrilaterals. This resulted in approximately 11 000 quads and 2000 remaining triangles. The final all-quad mesh, obtained after global h-refinement, had approximately 50 000 elements. The generation process is shown in Figs 20(b–d). Close-ups of the final mesh, shown in Figs 20(e–f), indicate that high quality surface grids may be obtained in this way.



(c)

Fig. 19. Chip simulation: (a) surface definition; (b) surface mesh; (c) simulation results ($Re = 1000$, streamlines).

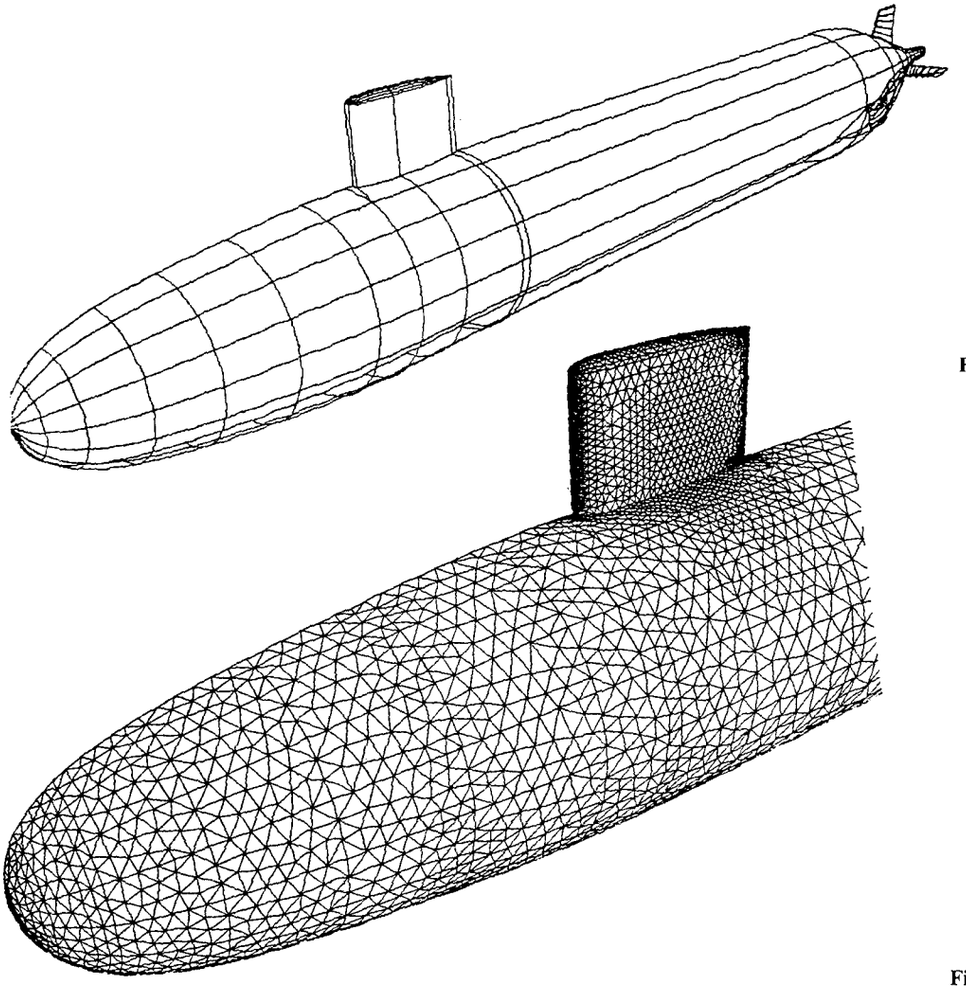


Fig. 20. (a).

Fig. 20. (b).

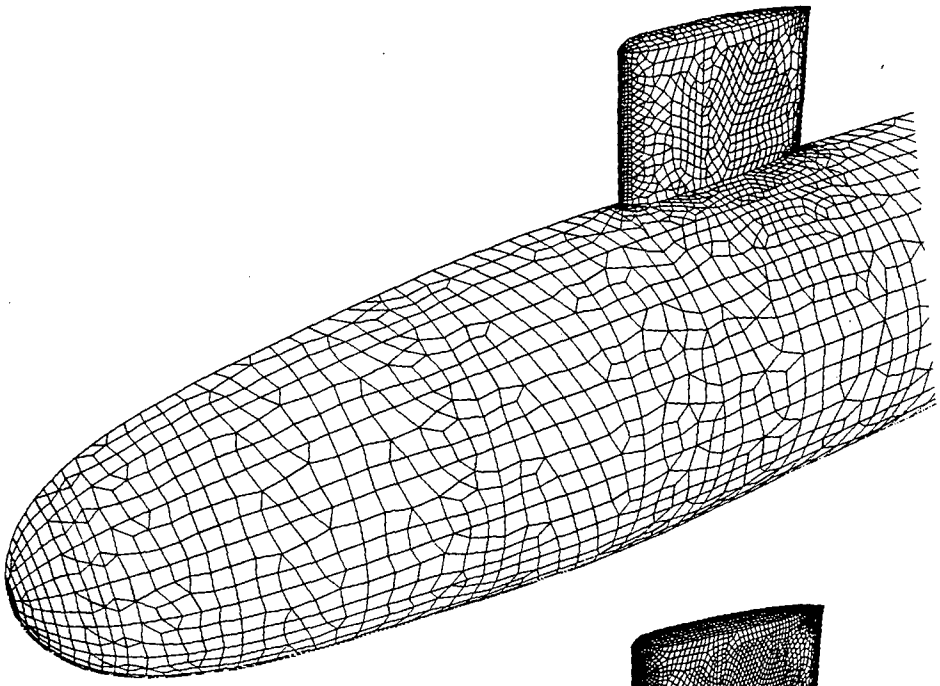


Fig. 20. (c).

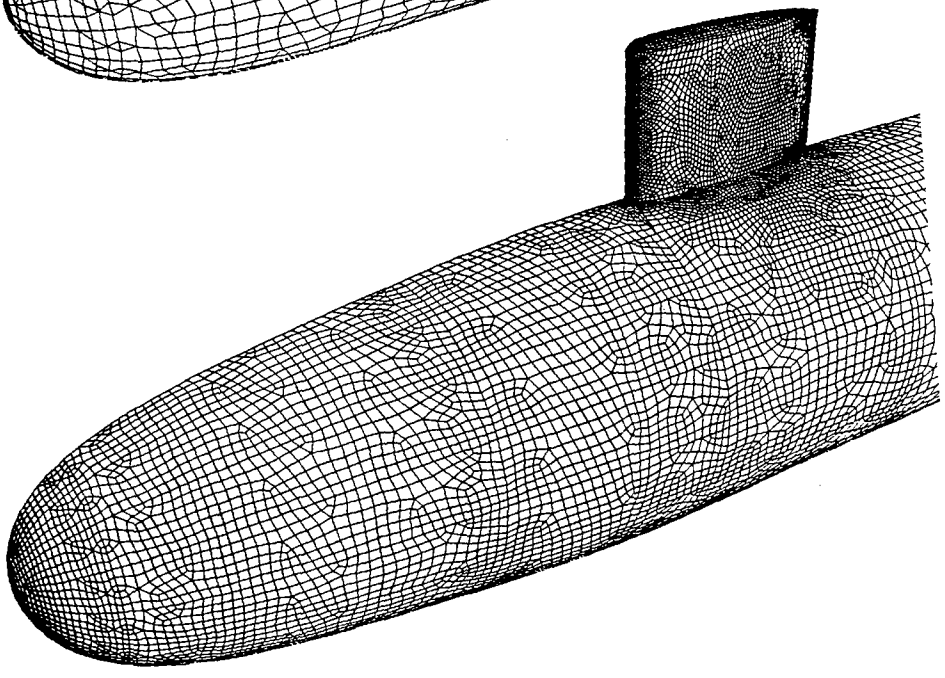


Fig. 20. (d).

6.5. Impacting Rod

Figure 21 shows the comparison of two runs conducted by using (a) a conventional hexahedral grid and (b) a hexahedral grid obtained by subdivision of tetrahedra with DYNA3D. Observe that the results are very similar, but that the mesh used for the 'brick-from-tet' mesh is actually finer. Given that a major portion of typical analysis man-hours is devoted to grid generation, we consider these results as very encouraging. Given the flexibility and degree of automation that tetrahedral grid generators have achieved, the splitting approach seems an attractive alternative to other hexahedral grid generation schemes for geometrically complex problem.

6.6. Definition of a Die Stamping Surface

This example shows how input preparation times may be reduced by working directly from discrete data instead of CAD data. The original CAD data set had over 500 surface patches, many of them overlapping and in need of trimming. The cloud of points, obtained from a digitization of the actual part, had 4398 points and is shown in Fig. 22(a). The final surface obtained is depicted in Fig. 22(b). Observe the presence of

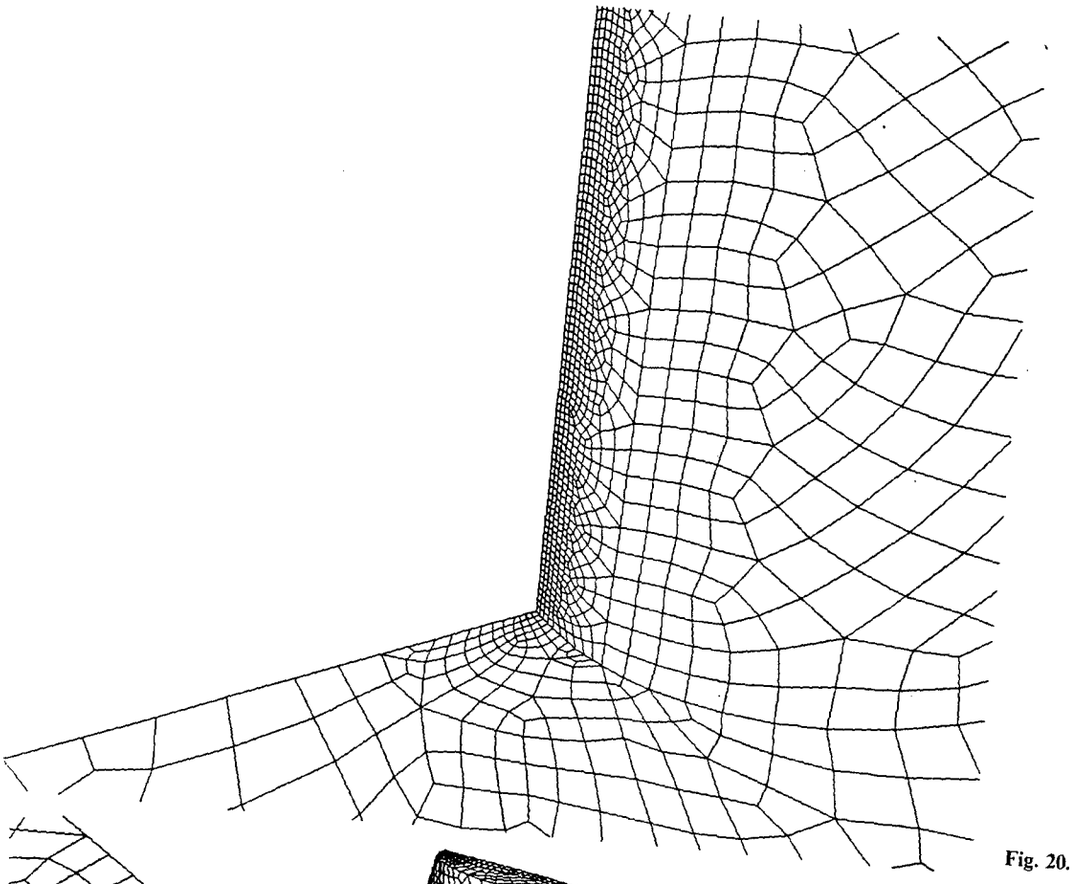
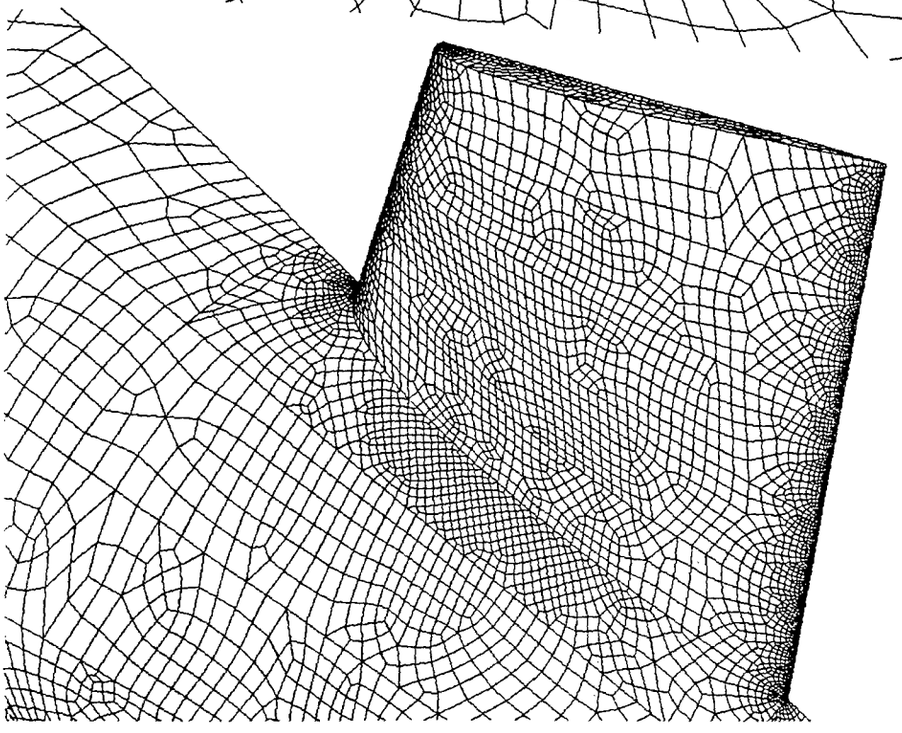


Fig. 20. (e).



(f)

Fig. 20. Generic submarine configuration: (a) surface definition; (b) surface mesh (all-tria); (c) surface mesh (mixed quad-tria); (d) surface mesh (all-quad); (e) close-up of tail region; (f) close-up of tail region.

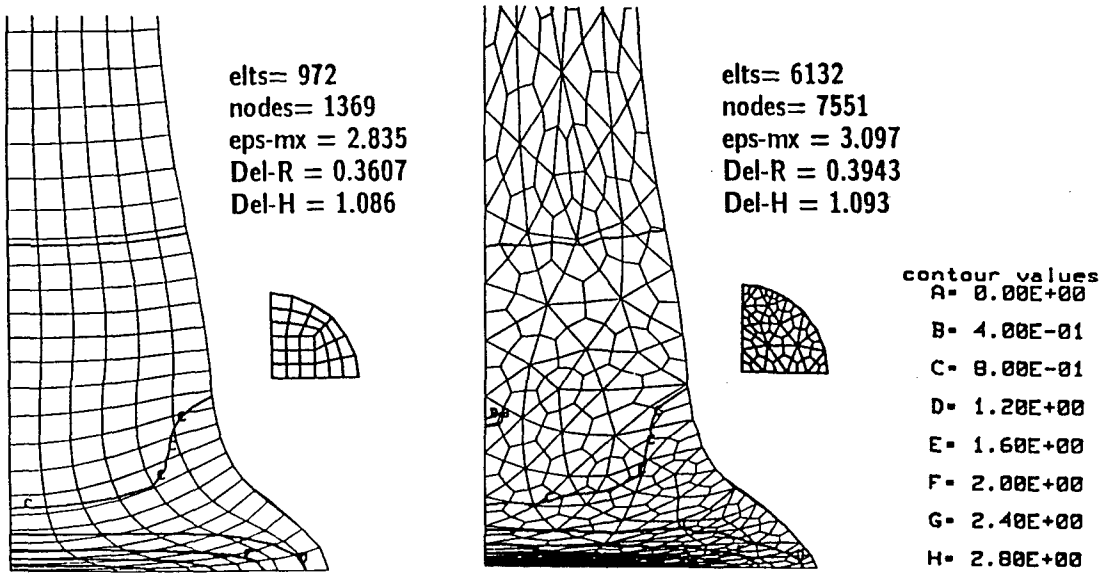


Fig. 21. Impacting rod problem: (a) standard 'good-looking' mesh results; (b) 'brick from tet' mesh results.

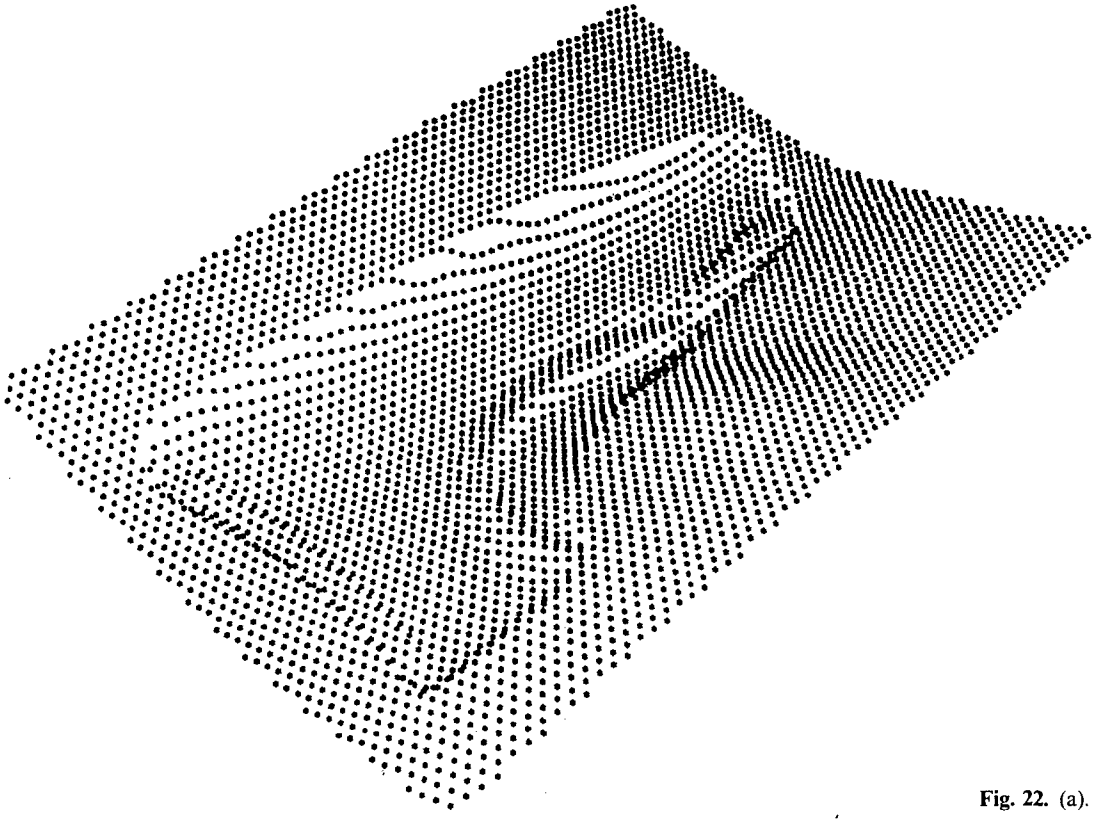
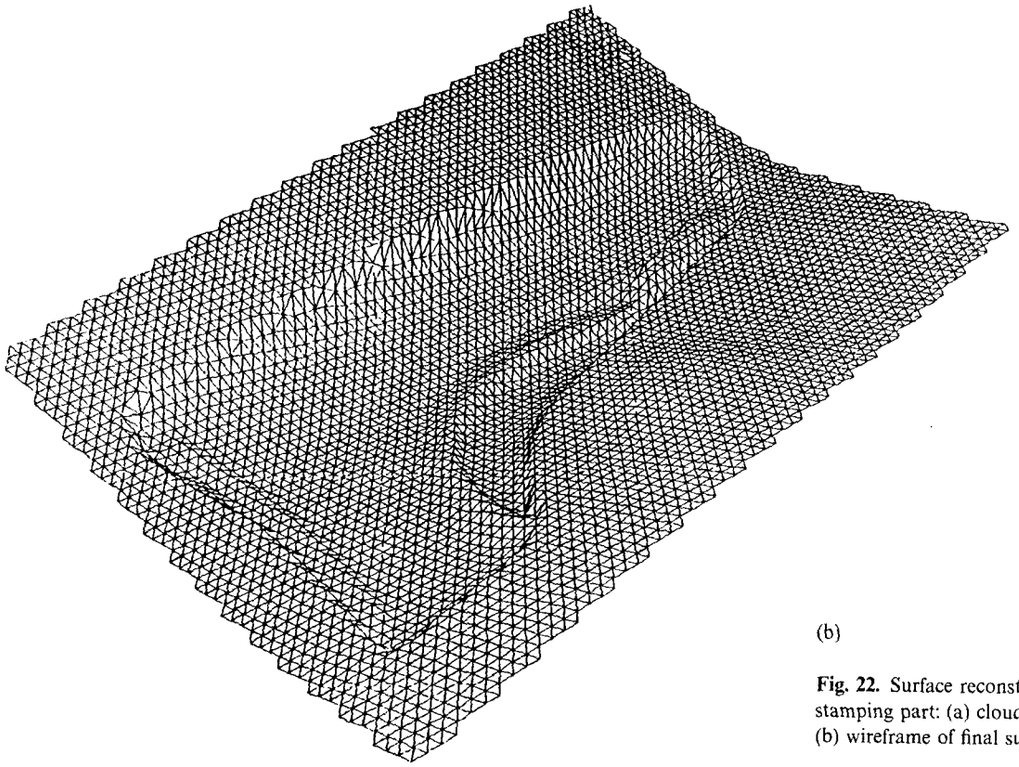


Fig. 22. (a).

ridges at the edge of the part. In some cases, the angles between adjacent faces exceeded 60°. The surface triangulation for this cloud of points took less than 30 s on the IBM/RS6000-550. The example clearly demonstrates the possible advantages of surface descriptions via discrete point sets. Trimming and combining over 500 surfaces is a tedious and time-consuming effort, which can be reduced drastically as shown here.

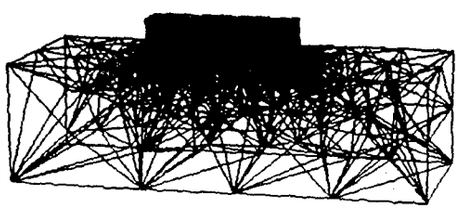
6.7. Parallel Grid Generation of a Double Missile Configuration

The mesh surrounding a double missile configuration in a bay was generated on the Intel Touchstone Delta

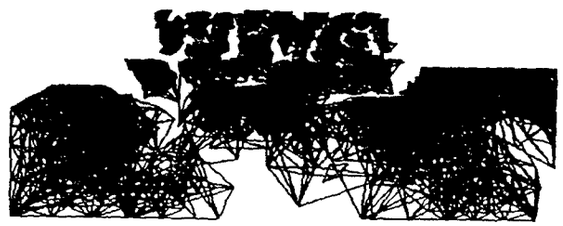


(b)

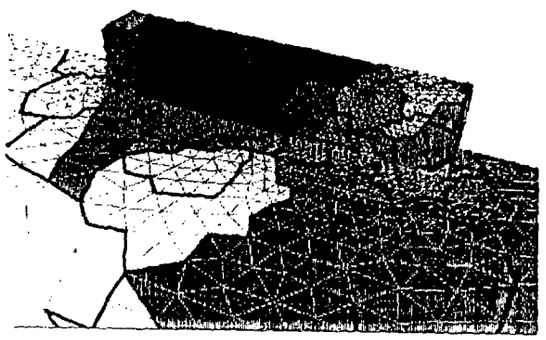
Fig. 22. Surface reconstruction for a die stamping part: (a) cloud of points; (b) wireframe of final surface.



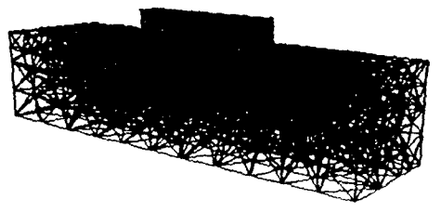
Background Mesh



Mesh After Subdomain Generation



Surface of Final Assembled Mesh



Final Mesh

Fig. 23. Parallel grid generation of a double missile configuration.

machine using 34 processors. The configuration is shown in Fig. 23. The partition of the background grid, as well as the elements obtained after the first pass over the processors, are shown in Fig. 23. After this stage, all the inter-subdomains regions are also meshed in parallel. The final mesh, consisting of 91 000 elements, is shown in Fig. 23.

7. Conclusions

This paper has summarized recent extensions and improvements to the advancing front grid generation technique. These improvements have considered the range of applicability, speed, and user-friendliness of this fast maturing technique. The range of applicability was enlarged by the ability to produce volumetric grids around thin surfaces (such as shells, membranes, fabrics, or surfaces with cusps), the generation of high aspect ratio grids for Navier–Stokes applications, the generation of higher order triangular and tetrahedral elements, and the generation of quadrilateral and hexahedral elements. Speed improvements were the result of reduced search overheads, as well as vectorization and parallelization. User-friendliness was enhanced by the development of direct gridding from discrete data, as well as simpler ways of specifying the desired element size and shape in space. The numerous examples included clearly demonstrate the versatility and maturity that advancing front grid generators have achieved. Further improvements in all of these areas may result from ongoing research. In particular, mesh smoothing and optimization [55] and the seamless integration to CAD systems represents topics that were not treated here but deserve increased attention in the future.

Acknowledgments

The author would like to acknowledge the help of Drs J. D. Baum and H. Luo for the World Trade Center configuration, as well as Drs D. Pelessone and Ch. Charman for the DYNA3D runs with quadrilateral and hexahedral grids. Dr A. Shostko coded the parallel grid generation procedure in 3-D, and his many innovations and improvements are gratefully acknowledged.

The work described on grid generation would have been impossible without the generous support of AFOSR and DNA over the years, as well as IBM and CRAY Research.

References

1. Baum, J.D.; Löhner, R. (1991) Numerical simulation of shock interaction with a modern main battlefield tank, AIAA-91-1666
2. Baum, J.D.; Luo, H.; Löhner, R. (1993) Numerical simulation of a blast inside a Boeing 747; AIAA-93-3091
3. Baum, J.D.; Luo, H.; Löhner, R. (1995) Numerical simulation of a blast inside the World Trade Center, AIAA-95-3091
4. van Phai, N. (1982) Automatic mesh generation with tetrahedron elements, International Journal for Numerical Methods in Engineering, 18, 237–289
5. Lo, S.H. (1985) A new mesh generation scheme for arbitrary planar domains, International Journal for Numerical Methods in Engineering, 21, 1403–1426
6. Peraire, J.; Vahdati, M.; Morgan, K.; Zienkiewicz, O.C. (1987) Adaptive remeshing for compressible flow computations, Journal of Computer Physics, 72, 449–466
7. Löhner, R. (1988) Some useful data structures for the generation of unstructured grids, Communications in Applied Numerical Methods, 4, 123–135
8. Löhner, R.; Parikh, P. (1988) Three-dimensional grid generation by the advancing front method, International Journal for Numerical Methods for Fluids, 8, 1135–1149
9. Peraire, J.; Morgan, K.; Peiro, J. (1990) Unstructured finite element mesh generation and adaptive procedures for CFD, AGARD-CP-46, 18
10. Löhner, R. (1992) Finite elements in CFD: grid generation, adaptivity and parallelization, Chapter 8 in AGARD Rep. 787, Proceedings Special Course on Unstructured Grid Methods for Advection Dominated Flows, VKI, Belgium, May and NASA Ames, Moffet Field, CA, September
11. Blacker, T.D.; Stephenson, M.B. (1992) Paving: a new approach to automated quadrilateral mesh generation, International Journal for Numerical Methods in Engineering, 32, 811–847
12. Blacker, T.D.; Meyers, R.J. (1993) Seams and wedges in plastering: a 3-D hexahedral mesh generation algorithm, Engineering with Computers, 9, 83–93
13. Baker, T.J. (1987) Three-dimensional mesh generation by triangulation of arbitrary point sets, AIAA-CP-87-1124, 8th CFD Conf., Hawaii
14. Baker, T.J. (1989) Developments and trends in three-dimensional mesh generation, Applied Numerical Mathematics 5, 275–304
15. Holmes, D.G.; Snyder, D.D. (1988) The generation of unstructured triangular meshes using Delaunay triangulation, Numerical Grid Generation in Computational Fluid Dynamics (Sengupta *et al.*, Editors), Pineridge Press, Swansea, 643–652
16. Mavriplis, D. (1990) Euler and Navier–Stokes computations for two-dimensional geometries using unstructured meshes, ICASE Rep. 90-3
17. Weatherill, N.P. (1992) Delaunay triangulation in computational fluid dynamics, Computer Mathematics Applications 24, 5/6, 129–150
18. Weatherill, N.P.; Hassan, O. (1994) Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints, International Journal for Numerical Methods in Engineering, 37, 2005–2039
19. Müller, J.-D. (1993) Proven angular bounds and stretched triangulations with the frontal Delaunay method, AIAA-93-3347-CP
20. Yerry, M.A.; Shepard, M.S. (1984) Automatic three-dimensional mesh generation by the modified-octree technique, International Journal for Numerical Methods in Engineering, 20, 1965–1990
21. Shepard, M.S.; Georges, M.K. (1991) Automatic three-dimensional mesh generation by the finite octree technique, International Journal for Numerical Methods in Enginering, 32, 709–749
22. Jin, H.; Tanner, R.I. (1993) Generation of unstructured tetrahedral meshes by the advancing front technique, Inter-

national Journal for Numerical Methods in Engineering 36, 1805–1823
23. Frykestig, J. (1990) Advancing front mesh generation techniques with application to the finite element method, Pub. 94:10, Chalmers University of Technology, Göteborg, Sweden
24. Bonet, J.; Peraire, J. (1991) An alternate digital tree algorithm for geometric searching and intersection problems, International Journal for Numerical Methods in Engineering, 31, 1–17.
25. George, P.L. (1991) Automatic Mesh Generation, John Wiley, New York
26. Luo, H.; Baum, J.D.; Löhner, R. (1994) Edge-based finite element scheme for the Euler equations, AIAA Journal 32, 6, 1183–1190
27. Mestreau, E.; Löhner, R. (1994) Numerical simulation of chip cooling via large-scale FEM simulations, CSI-GMU Preprint
28. Löhner, R. (1989) Adaptive remeshing for transient problems, Computer Methods in Applied Mechanics and Engineering, 75, 195–214
29. Löhner, R. (1990) Three dimensional fluid-structure interaction using a finite element solver and adaptive remeshing, Computer Systems in Engineering, 1, 2–4, 257–272
30. Tilch, R. (1991) PhD Thesis, CERFACS, Toulouse, France
31. Peraire, J.; Peiro, J.; Morgan, K. (1992) Adaptive remeshing for three-dimensional compressible flow computations, Journal of Computer Physics 103, 269–285
32. Rank, E.; Schweingruber, M; Sommer, M. (1993) Adaptive mesh generation and transformation of triangular to quadrilateral meshes, Communications in Applied Numerical Methods, 9, 121–129
33. Nakahashi, K. (1987) FDM-FEM zonal approach for viscous flow computations over multiple bodies, AIAA-87-0604
34. Nakahashi, K.; Obayashi, S. (1987) Viscous flow computations using a composite grid, AIAA-CP-87-1128, 8th CFD Conf., Hawaii
35. Nakahashi, K. (1988) Optimum spacing control of the marching grid generation, AIAA-88-0515
36. Kallinderis, Y.; Ward, S. (1992) Prismatic grid generation with an efficient algebraic method for aircraft configurations, AIAA-92-2721
37. Pirzadeh, S. (1993) Unstructured viscous grid generation by advancing-layers method, AIAA-93-3453
38. Pirzadeh, S. (1994) Viscous unstructured three-dimensional grids by the advancing-layers method, AIAA-94-0417
39. Morgan, K.; Probert, J.; Peraire, J. (1993) Line relaxation methods for the solution of two-dimensional and three-dimensional compressible flows, AIAA-93-3366
40. Hestenes, M.; Stiefel, E. (1952) Methods of conjugate gradients for solving linear systems, Journal of National Bureau of Standards, 49, 409–436
41. Gentzsch, W.; Schlüter, A. (1978) Über ein Einschrittverfahren mit zyklischer Schrittweitenänderung zur Lösung parabolischer Differentialgleichungen, ZAMM, 58, T415–T416
42. Löhner, R.; Morgan, K. (1987) An unstructured multigrid method for elliptic problems, International Journal for Numerical Methods in Engineering, 24, 101–115
43. Marchant, M.J.; Weatherill, N.P. (1993) The construction of nearly orthogonal multiblock grids for compressible flow simulation, Communications in Applied Numerical Methods, 9, 567–578
44. Löhner, R. (1993) Matching semi-structured and unstructured grids for Navier–Stokes calculations, AIAA-93-3348-CP
45. Zienkiewicz, O.C.; Taylor, R. (1988) The Finite Element Method, McGraw Hill, New York
46. Goudreau, G.L.; Hallquist, J.O. (1982) Recent developments in large-scale finite element lagrangean hydrocode technology, Computer Methods in Applied Mechanics and Engineering, 33, 725–757
47. Whirley, R.G.; Hallquist, J.O. (1991) DYNA3D, a nonlinear explicit, three-dimensional finite element code for solid and structural mechanics—User Manual, UCRL-MA-107254
48. Löhner, R.; Yang, C.; Cebral, J.; Baum, J.; Luo, H.; Charman, C.; Pelessone, D. (1994) A loose coupling algorithm for fluid-structure interaction simulations; Proceedings 8th Annual Idaho National Engineering Lab. Comp. Symp. 10-3, October
49. Löhner, R.; Camberos, J.; Merriam, M. (1992) Parallel unstructured grid generation, Computer Methods in Applied Mechanics and Engineering, 95, 343–357
50. Shotsko, A.; Löhner, R. (1994) Three-dimensional parallel unstructured grid generation, AIAA-94-0418
51. Choi, B.K.; Chin, H.Y.; Loon, Y.I.; Lee, J.W. (1988) Triangulation of scattered data in 3D space, Computer Aided Geometric Design, 20, 239–248
52. Hoppe, H.; DeRose, T.; Duchamp, T.; McDonald, J.; Stuetzle, W. (1992) Surface reconstruction from unorganized points, Computer Graphics 26, 2, 71–78
53. Hoppe, H.; DeRose, T.; Duchamp, T.; McDonald, J.; Stuetzle, W. (1993) Mesh optimization, Proceedings Computer Graphics Annual Conference, 19–26
54. Löhner, R. (1994) Surface reconstruction from clouds of points, CSI-GMU Preprint
55. Cabello, J.; Lönher, R.; Jacquotte, O-P. (1992) A variational method for the optimization of two- and three-dimensional unstructured meshes, AIAA-92-0450